

# 主要内容

---

- 格式化Swap设备及挂载
- 实现Swap.c相关函数，给Linux0.11增加更多虚拟地址功能
- 测试程序

## 一，格式化Swap设备及挂载

---

### 1.格式化Swap设备

- a. 第一个扇区用作MBR分区表，且第一个扇区最后有Magic Number: 55aa，表示分区有效
- b. 接下来一页用作bit\_map。Swap分区一共32MB，8092页。对应8092位，即1024个字节
- c. 该页最后四个字节为签名SWAP

MBR由三部分构成：

1. 主引导程序代码，占446字节
2. 硬盘分区表DPT，占64字节，最多支持4个分区
3. 主引导扇区结束标志AA55H

标准 MBR 结构

地址			描述		长度 (字节)
Hex	Oct	Dec			
0000	0000	0	代码区		440 (最大 446)
01B8	0670	440	选用软盘标志		4
01BC	0674	444	一般为空值: 0x0000		2
01BE	0676	446	标准 MBR 分区表规划 (四个 16 byte的主分区表入口)		64
01FE	0776	510	55h	MBR 有效标志: 0xAA55	2
01FF	0777	511	AAh		
MBR, 总大小: 446 + 64 + 2 =					512

表1 图2分区表第一字段			
字节位移	字段长度	值	字段名和定义
0x01BE	BYTE	0x80	引导指示符(Boot Indicator) 指明该分区是否是活动分区。
0x01BF	BYTE	0x01	开始磁头(Starting Head)
0x01C0	6位	0x01	开始扇区(Starting Sector) 只用了0~5位。后面的两位(第6位和第7位)被开始柱面字段所使用
0x01C1	10位	0x00	开始柱面(Starting Cylinder) 除了开始扇区字段的最后两位外, 还使用了1位来组成该柱面值。开始柱面是一个10位数, 最大值为1023
0x01C2	BYTE	0x07	系统ID(System ID) 定义了分区的类型, 详细定义, 请参阅图4
0x01C3	BYTE	0xFE	结束磁头(Ending Head)
0x01C4	6位	0xFF	结束扇区(Ending Sector) 只使用了0~5位。最后两位(第6、7位)被结束柱面字段所使用
0x01C5	10位	0x7B	结束柱面(Ending Cylinder) 除了结束扇区字段最后的两位外, 还使用了1位, 以组成该柱面值。结束柱面是一个10位的数, 最大值为1023
0x01C6	DWORD	0x0000003F	相对扇区数(Relative Sectors) 从该磁盘的开始到该分区的开始的位移量, 以扇区来计算
0x01CA	DWORD	0x00DAA83D	总扇区数(Total Sectors) 该分区中的扇区总数

这里使用新添加的一块硬盘作为swap分区。

这块硬盘总大小64MB，这里只分了一个32MB的分区，作为Swap分区。

从下图的MBR表中可以看出，这一分区开始于0磁头，3扇区，0柱面。82标识Linux Swap分区。结束于15磁头，38扇区，107柱面。共1538107-2=65662。

逻辑上始于第2扇区，总共65662扇区。

```
micz@ubuntu:~/oslab2/workbench$ hexdump newdisk.img -C -s 432
000001b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000001c0  03 00 82 0f 26 6b 02 00 00 00 7e 00 01 00 00 00 | ....&k....~....|
000001d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
000001f0  00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa | .....U.|
00000200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
00000400  fe ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....|
00000410  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....|
*
00000800  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
000013f0  00 00 00 00 00 00 00 00 00 00 00 00 53 57 41 50 | .....SWAP|
00001400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
03c90000
```

0x1fe处还有Magic Number: 0xaa55, 表示MBR有效

从0x400开始, 也就是第1kb处, 即第三个扇区, 是bitmap。

可以看出第1位已被标识成0, 表示第一页已作它用, 不能参与swap。

之后直到0x800都为1, 表示可用, 正好是1024字节, 即8096位, 对应8096个页面, 也就是32MB的Swap分区。

超出32MB不可用, 因此0x800之后到这一页结束, 都置为0。

```
micz@ubuntu:~/oslab2/workbench$ hexdump newdisk.img -C -s 432
000001b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000001c0  03 00 82 0f 26 6b 02 00 00 00 7e 00 01 00 00 00 | ....&k....~....|
000001d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
000001f0  00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa | .....U.|
00000200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
00000400  fe ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....|
00000410  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....|
*
00000800  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
000013f0  00 00 00 00 00 00 00 00 00 00 00 00 53 57 41 50 | .....SWAP|
00001400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
03c90000
```

注意在第一页的结尾有一个签名, SWAP, 用来标识这个Swap设备。

## 2.Swap设备挂载到Bochs

在 bochsrc.bxrc 中添加

```
ata0-slave: type=disk, path="$OS_PATH/newdisk.img", mode=flat, cylinders=204, heads=16,
spt=38
```

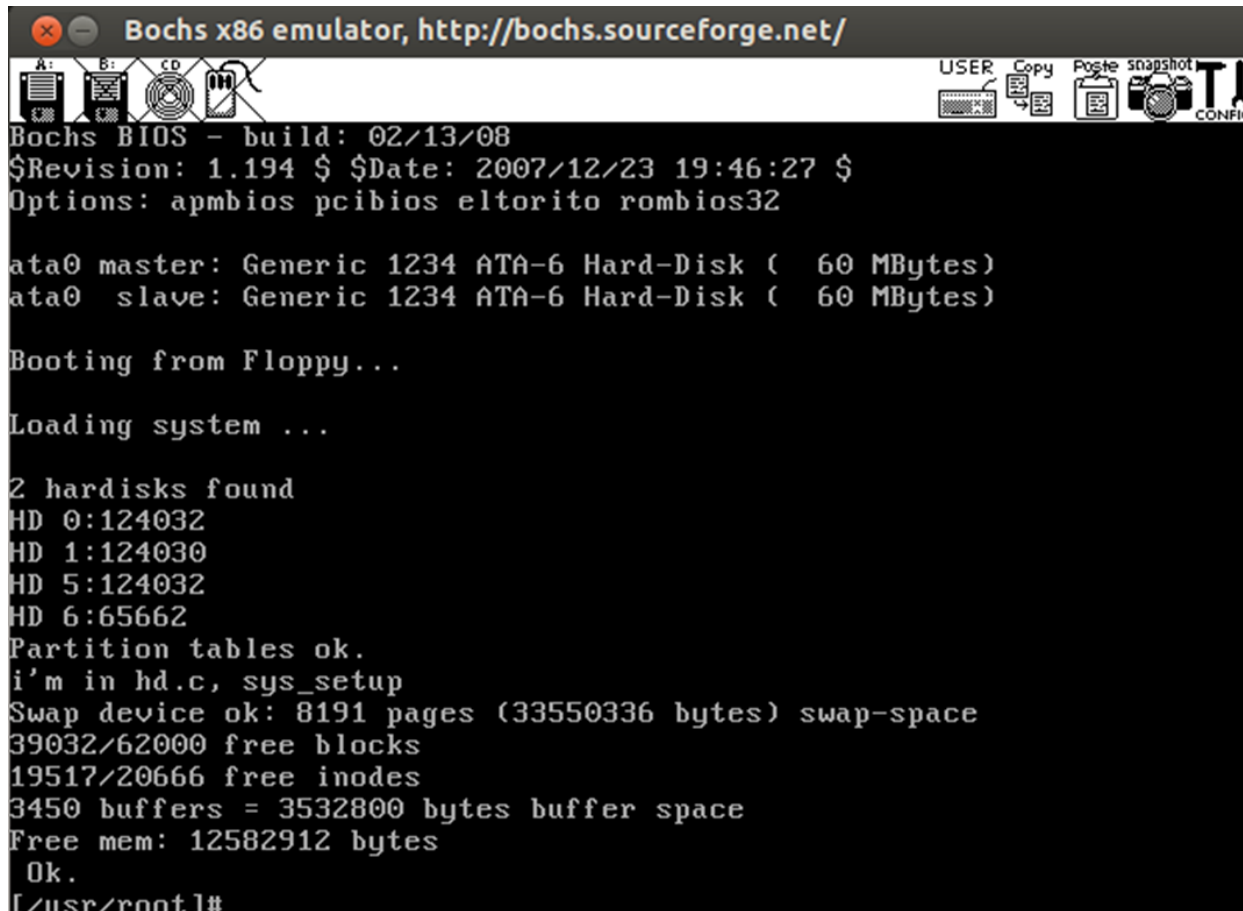
$204 * 16 * 38 * 512 = 63.5M$ (注意这里1MB=1000KB, 生成.img文件时)

## 二, Swap.c相关函数

# 1.Linux初始化时读取Swap设备信息并输出

在hd.c函数里面的setup系统调用可以读取系统挂载的所有硬盘的信息。输出效果如下图：

读取信息并输出结束后，会调用init\_swapping函数。



```
Bochs BIOS - build: 02/13/08
$Revision: 1.194 $ $Date: 2007/12/23 19:46:27 $
Options: apmbios pcibios eltorito rombios32

ata0 master: Generic 1234 ATA-6 Hard-Disk ( 60 MBytes)
ata0 slave: Generic 1234 ATA-6 Hard-Disk ( 60 MBytes)

Booting from Floppy...

Loading system ...

2 harddisks found
HD 0:124032
HD 1:124030
HD 5:124032
HD 6:65662
Partition tables ok.
i'm in hd.c, sys_setup
Swap device ok: 8191 pages (33550336 bytes) swap-space
39032/62000 free blocks
19517/20666 free inodes
3450 buffers = 3532800 bytes buffer space
Free mem: 12582912 bytes
Ok.
[/usr/root]#
```

## 2.Init\_Swapping函数

输出所有硬盘的大小及分区后，就要执行swap分区的初始化。即读取swap分区的大小，看其大小是否符合规范。还要从swap分区第一页获得bitmap，并检查其是否合法

bitmap所在页需要满足以下条件：

1. 页的最后四个字节为SWAP;
2. bitmap的第一位为0，即不可用(被bitmap占用了);
3. 超过swap\_size的那些位应为0，即不可用。比如这里swap分区只有32M，对应8092个页，只需要8092位，也就是1024字节去表示。还剩3072字节，应该全置为0，防止访问到32MB之外。

最后，该函数要统计一下swap分区可用的页数有多少。如果可用的页数为0，则该swap分区已不可用。

注意这里我们采用第二块硬盘的第一个分区作为swap分区，因此SWAP\_DEV=0x306

硬盘逻辑设备号	逻辑设备号	对应设备文件
0x300	/dev/hd0	代表整个第1个硬盘
0x301	/dev/hd1	表示第1个硬盘的第1个分区
0x302	/dev/hd2	表示第1个硬盘的第2个分区
0x303	/dev/hd3	表示第1个硬盘的第3个分区
0x304	/dev/hd4	表示第1个硬盘的第4个分区
0x305	/dev/hd5	代表整个第2个硬盘
0x306	/dev/hd6	表示第2个硬盘的第1个分区
0x307	/dev/hd7	表示第2个硬盘的第2个分区
0x308	/dev/hd8	表示第2个硬盘的第3个分区
0x309	/dev/hd9	表示第2个硬盘的第4个分区

对于硬盘这类块设备，Linux0.11中只实现了ll\_rw\_block()，即按块读写的函数

对于Swap分区的读写是按页进行的，为了方便我们要先实现一个ll\_rw\_page()函数。即每次访问作为一个request，添加到硬盘队列中去，这个request每次读8个扇区，即4KB

```
req->dev = dev;
req->cmd = rw;
req->errors = 0;
req->sector = page<<3;
req->nr_sectors = 8;
req->buffer = buffer;
req->waiting = current;
req->bh = NULL;
req->next = NULL;
```

对于swap设备，它的设备号是固定的，因此我们可以用宏专门来进行swap设备的读写，减少参数



```
#define read_swap_page(nr,buffer) ll_rw_page(READ,SWAP_DEV,(nr),(buffer));
#define write_swap_page(nr,buffer) ll_rw_page(WRITE,SWAP_DEV,(nr),(buffer));
```

- get\_swap\_page()  
扫描bitmap, 若找到有一位是1, 就把它置为不可用, 并返回它的序号, 表示找到了这一页free page
- swap\_free(int nr)  
检查第nr位是否是0, 如果是, 就置为1并返回。否则报错。
- swap\_in()  
从swap设备调一页到内存中
- swap\_out()  
从将内存中的一页移出到swap设备中

## Swap out()

该函数在get\_free\_page()中被调用。原来0.11中, get\_free\_page()是从16MB物理空间的尾部开始查找空闲的物理页。如果没有空闲的就返回0。但是这里有了swap功能, 如果扫描一遍没找到空闲物理页, 就要调用swap\_out()函数, 然后再次申请页面。

```
1  int swap_out(void)
2  {
3      static int dir_entry = FIRST_VM_PAGE>>10;
4      static int page_entry = -1;
5      int counter = VM_PAGES;
6      int pg_table;
7
8      while (counter>0) {
9          pg_table = pg_dir[dir_entry];
10         if (pg_table & 1)
11             break;
12         counter -= 1024;
13         dir_entry++;
14         if (dir_entry >= 1024)
15             dir_entry = FIRST_VM_PAGE>>10;
16     }
17     pg_table &= 0xffff000;
18     while (counter-- > 0) {
19         page_entry++;
20         if (page_entry >= 1024) {
21             page_entry = 0;
22         repeat:
23             dir_entry++;
24             if (dir_entry >= 1024)
25                 dir_entry = FIRST_VM_PAGE>>10;
26             pg_table = pg_dir[dir_entry];
27             if (!(pg_table&1))
28                 if ((counter -= 1024) > 0)
29                     goto repeat;
30             else
```

```

31         break;
32         pg_table &= 0xfffff000;
33     }
34     if (try_to_swap_out(page_entry + (unsigned long *) pg_table)){
35         printk("I'm swapping out at %u\n\r",page_entry + (unsigned long *)
pg_table);
36         return 1;
37     }
38
39 }
40 printk("Out of swap-memory\n\r");
41 return 0;
42 }

```

查找可被换出的页面过程如下：

从线性空间中的任务1的第一页开始往后找查找页目录，找最低位为1的项->查找页表，找到最低位为1的项->尝试换出try\_to\_swap\_out()。若无法换出，继续查找如果成功换出，则将swap\_nr左移1位存放到table\_ptr中，以备将来使用

### Try\_to\_swap\_out():

在swap\_out()函数中，找到的只是有可能被换出的页(有对应物理内存的页)。但这些页未必真的能换出，try\_to\_swap\_out()还会做进一步检查

- 再次检查page最低位是否为1
- page大小是否越界
- 如果该页未经修改过，则直接释放，而不用往swap设备上写
- 如果该内存页未被使用(异常)，或无法获得swap\_nr(swap设备已满)，则不能交换

```

1  int try_to_swap_out(unsigned long * table_ptr)
2  {
3      unsigned long page;
4      unsigned long swap_nr;
5
6      page = *table_ptr;
7      if (!(PAGE_PRESENT & page)) //page地址最右如果为1，说明是一个有效页表项(有对应的物理
内存)。0x01&page
8          return 0;
9      if (page - LOW_MEM > PAGING_MEMORY)
10         return 0;
11     if (PAGE_DIRTY & page) { //只有当这个页面是被修改过时，才需要被换出。
12                                     //如果这个页面没被修改过，说明它肯定是从别处复制过来的，肯定
还有副本。
13                                     //这时就不必换出，直接使用即可。
14         page &= 0xfffff000;
15         if (mem_map[MAP_NR(page)] != 1)
16             return 0;
17         if (!(swap_nr = get_swap_page()))
18             return 0;

```



```

19     *table_ptr = swap_nr<<1; //swap_nr左移一位，使最右位为0。这样table_ptr最低位
    也为0，
20                                     //系统就会知道这一页表项是没有对应物理内存的。
21                                     //*table_ptr中储存的是原来那页在swap分区中的编号
    *2。
22     invalidate();
23     write_swap_page(swap_nr, (char *) page);
24     free_page(page);
25     return 1;
26 }
27 *table_ptr = 0;
28 invalidate();
29 free_page(page);
30 return 1;
31 }

```

### Swap\_in():

在发生缺页中断时被调用。Swap\_out()时已将该页在swap设备上的页码储存在table\_ptr中。通过它可以获得swap\_nr，通过read\_swap\_page()将其读回内存。读回后，要修改对应的bitmap位。还要修改table\_ptr成新的物理内存地址，并将标志位置为已修改过。

## 三，测试程序

不断malloc申请内存，并往内存中写入任意内容(如果不写的话，由于写时复制机制，系统不会真的分配物理内存)

在原始的Linux0.11系统上，申请2960页内存会发生oom错误

The screenshot shows a Bochs x86 emulator window with the title "Bochs x86 emulator, http://bochs.sourceforge.net/". The window contains a terminal window with the following text:

```

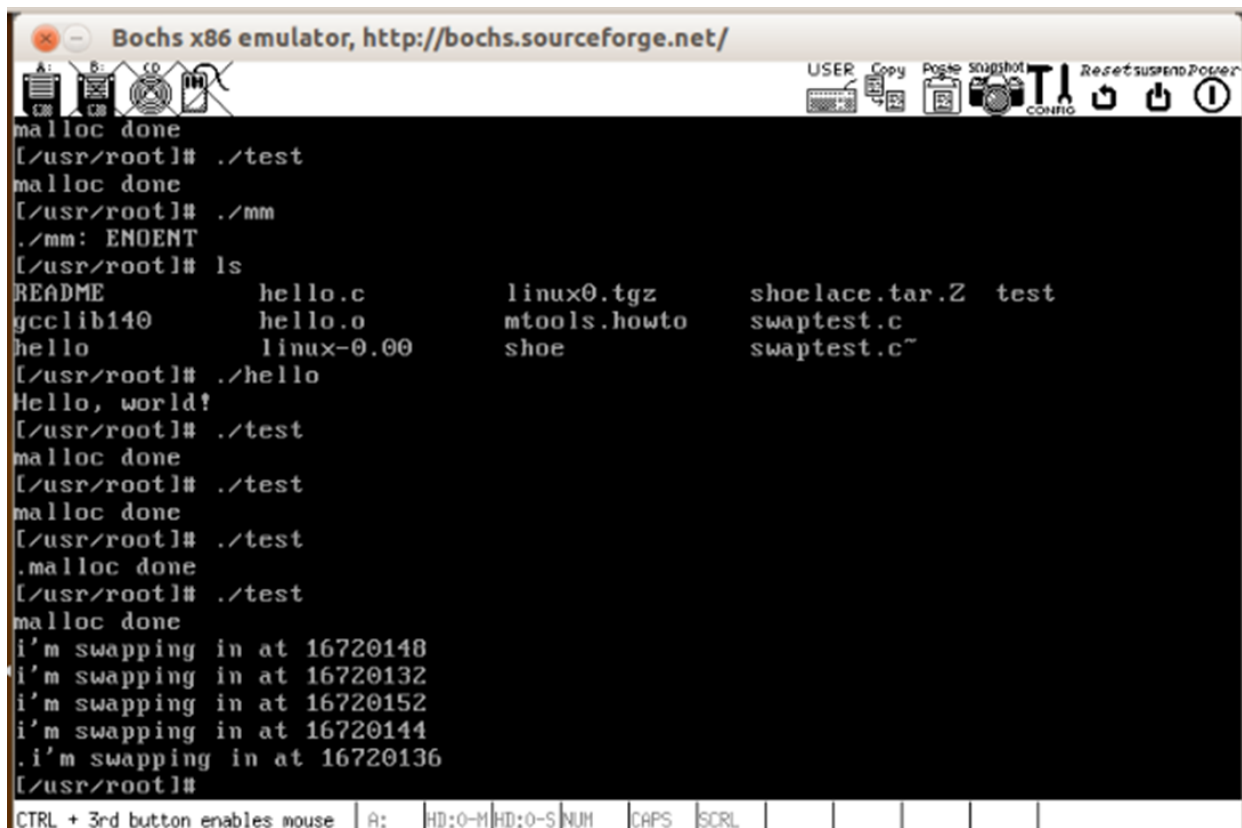
Bochs UBE Display Adapter enabled
Bochs BIOS - build: 02/13/08
$Revision: 1.194 $ $Date: 2007/12/23 19:46:27 $
Options: apmbios pcibios eltorito rombios32
ata0 master: Generic 1234 ATA-6 Hard-Disk ( 60 MBytes)
Booting from Floppy...
Loading system ...
Partition table ok.
39063/62000 free blocks
19518/20666 free inodes
3454 buffers = 3536896 bytes buffer space
Free mem: 12582912 bytes
Ok.
[/usr/rootl# gcc -o test swapttest.c
swapttest.c: In function main:
swapttest.c:6: warning: assignment of pointer from integer lacks a cast
./[/usr/rootl# ./test
out of memory
Segmentation fault
[/usr/rootl#

```

The bottom of the window shows a status bar with the text "CTRL + 3rd button enables mouse" and a row of icons for various functions like A:, HD:0-M, NUM, CAPS, SCRL, etc.

添加Swap后，运行效果如下：申请了2960页，超出16MB，发生换出操作，在执行换出操作时打印了待换出的物理内存地址

```
[/usr/root]# gcc -o test swapttest.c
swapttest.c: In function main:
swapttest.c:6: warning: assignment of pointer from integer lacks a cast
[/usr/root]# ./test
I'm swapping out at 16769144
I'm swapping out at 16769156
I'm swapping out at 16719872
I'm swapping out at 16719876
I'm swapping out at 16719880
I'm swapping out at 16719884
I'm swapping out at 16719888
I'm swapping out at 16719892
I'm swapping out at 16719896
I'm swapping out at 16719900
I'm swapping out at 16719904
I'm swapping out at 16719908
I'm swapping out at 16719912
I'm swapping out at 16719916
malloc done
```



```
Bochs x86 emulator, http://bochs.sourceforge.net/
USER Copy Paste Snapshot CTRL+ALT+R Reset suspend Power
malloc done
[/usr/root]# ./test
malloc done
[/usr/root]# ./mm
./mm: ENOENT
[/usr/root]# ls
README          hello.c          linux0.tgz       shoelace.tar.Z   test
gcclib140       hello.o          mtools.howto    swapttest.c
hello           linux-0.00       shoe             swapttest.c~
[/usr/root]# ./hello
Hello, world!
[/usr/root]# ./test
malloc done
[/usr/root]# ./test
malloc done
[/usr/root]# ./test
malloc done
[/usr/root]# ./test
malloc done
i'm swapping in at 16720148
i'm swapping in at 16720132
i'm swapping in at 16720152
i'm swapping in at 16720144
i'm swapping in at 16720136
[/usr/root]#
```

CTRL + 3rd button enables mouse | A: | HD:0-M | HD:0-S | NUM | CAPS | SCRL |

一次换入过多，会发生奇怪的错误

Bochs x86 emulator, <http://bochs.sourceforge.net/>

USER Copy Paste Snapshot CONFIG Reset suspend Power

```
[usr/root]# ./test
malloc done
[usr/root]# ./test
malloc done
i'm swapping in at 16720148
i'm swapping in at 16720132
i'm swapping in at 16720152
i'm swapping in at 16720144
i'm swapping in at 16720136
[usr/root]# ./test
malloc done
i'm swapping in at 16720172
i'm swapping in at 16720156
i'm swapping in at 16720160
i'm swapping in at 16720164
i'm swapping in at 16720168
[usr/root]# ./test i'm swapping in at 16720180
st
i'm swapping in at 16720176
malloc done
block on free list clobbered
Tell root to fix this someday.
Stopping myself...block on free list clobbered
Tell root to fix this someday.
Stopping myself...[usr/root]#
```