

Day 18

我们要丰富一下我们的命令行工具，让他成为一个真正可用的命令行工具。

首先我们发现我们的多个窗口的光标是同时闪烁的，这样是不符合我们平常使用的系统的特性的，也不美观，所以我们只允许获得焦点的窗体上的光标闪烁。

我们先从harimain开始修改。

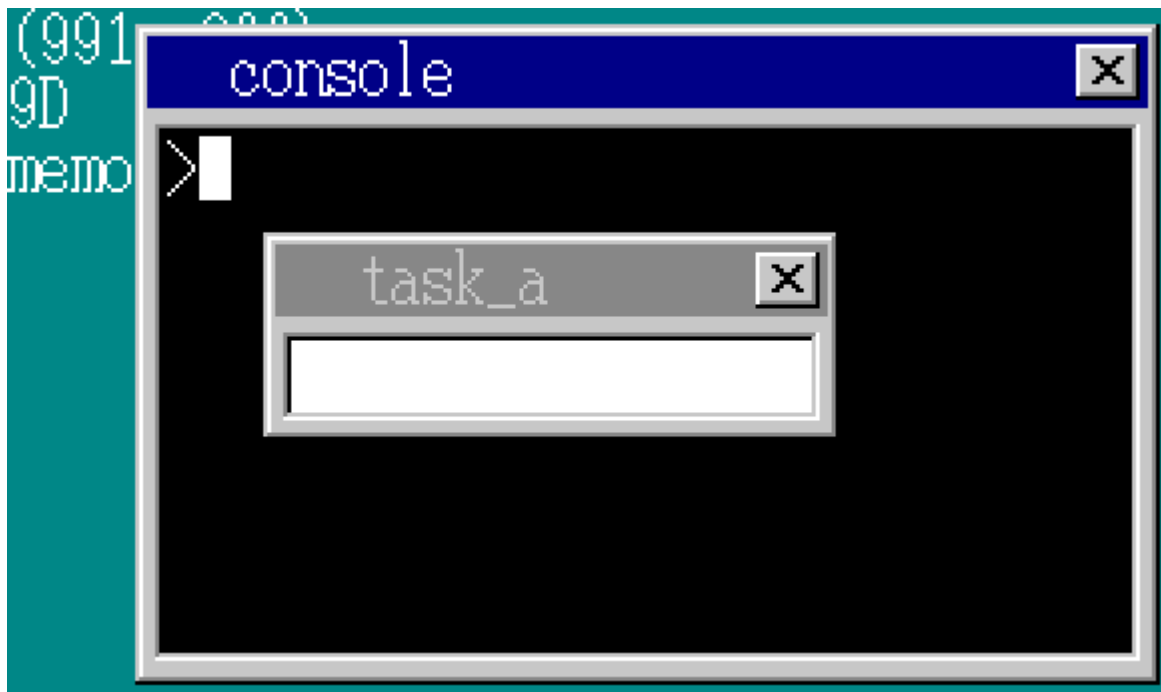
我们的主要思路是利用光标颜色的负数范围，来表明我们不想打印光标。

换句话说，如果我们的cursor_c是-1，那么我们就不打印光标。

除此之外，我们还要注意，在我们切换打印光标的颜色的时候，一定要先判断一下是否大于等于0，否则我们的-1就会被改掉。

另一种比较直观的思路是另设一个变量，来标记我们是否需要打印光标。作者的做法好处在于我们少开了一个变量，充分利用现有变量的值域空间，节省内存。

```
if (i == 256 + 0x0f) { /* Tab键*/
    if (key_to == 0) {
        key_to = 1;
        make_wtitle8(buf_win, sht_win->bysize, "task_a", 0);
        make_wtitle8(buf_cons, sht_cons->bysize, "console", 1);
        cursor_c = -1; /* 不显示光标 */
        boxfill8(sht_win->buf, sht_win->bysize, COL8_FFFFFFFF, cursor_x, 28, cursor_x + 7,
43);
    } else {
        key_to = 0;
        make_wtitle8(buf_win, sht_win->bysize, "task_a", 1);
        make_wtitle8(buf_cons, sht_cons->bysize, "console", 0);
        cursor_c = COL8_000000; /*显示光标*/
    }
    sheet_refresh(sht_win, 0, 0, sht_win->bysize, 21);
    sheet_refresh(sht_cons, 0, 0, sht_cons->bysize, 21);
}
```



可以看到，当焦点在console上时只有一个光标在闪烁，两个光标不会同时闪烁了。

而当焦点不再console上的时候，还是由两个光标在闪烁，这是因为我们目前只修改了harimain，也就是task_a。

我们接着来修改task_cons也就是console。

由于两个task是相对独立的，要修改task_b的cursor_c，就需要想别的办法。我们可以通过fifo将我们想要灭掉光标的信息发送过去我们先将光标开始闪烁定义为2，停止闪烁定义为3。

我们首先需要在harimain对 `Tab` 的处理稍稍修改一下，在修改完自己的cursor_c后，向task_cons.fifo中put一个2或者3。

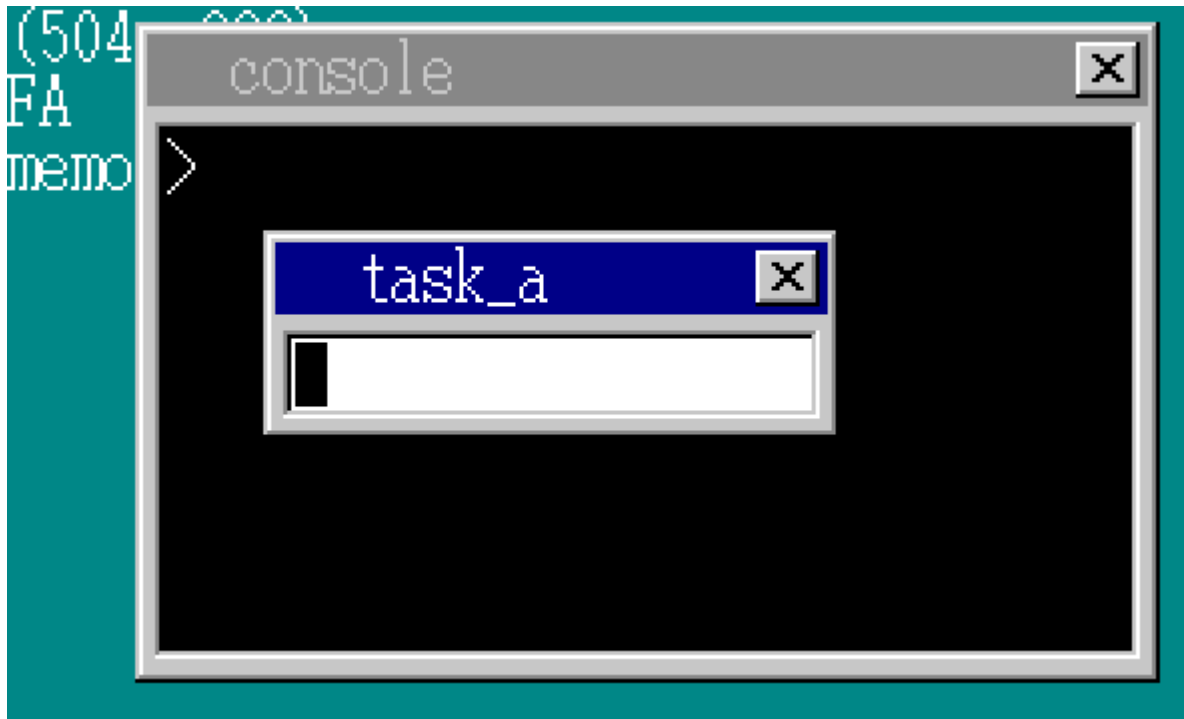
```
cursor_c = -1;
fifo32_put(&task_cons->fifo, 2);
// -----
cursor_c = COL8_000000;
fifo32_put(&task_cons->fifo, 3);
```

对了！还有初始状态！由于任意时刻只能由一个窗体有焦点，所以启动状态也是这样。我们要把task_cons的cursor_c的初始值设置成-1

```
int i, fifobuf[128], cursor_x = 16, cursor_c = -1;
```

就像这样

`make run` 一下



成功了，这次任意时刻只有一个窗体的光标在闪烁了

处理回车键，我们先只让回车键完成换行吧。当harimain捕获到回车且console具有焦点，我们向命令行窗口发送10+256（换行的ASCII是10）

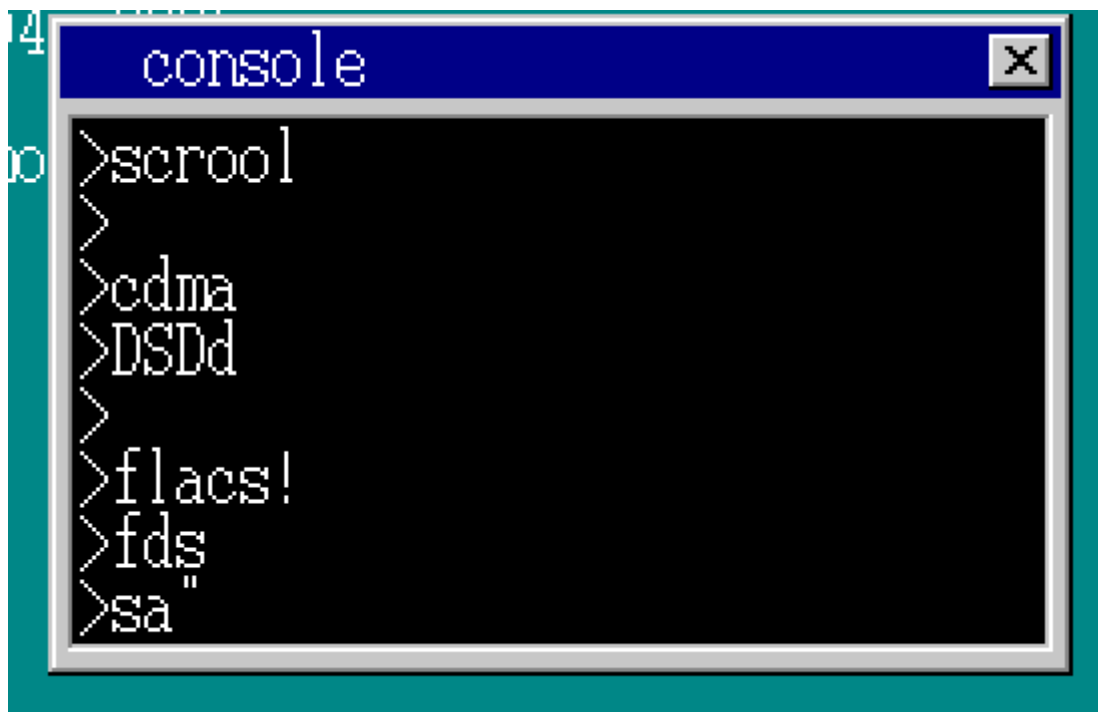
我们想下task_cons应该怎么修改，要完成换行，我们先要把旧光标删除，然后把光标（及输入位置）移到下一行的开始位置。打印一个提示符，打印光标。

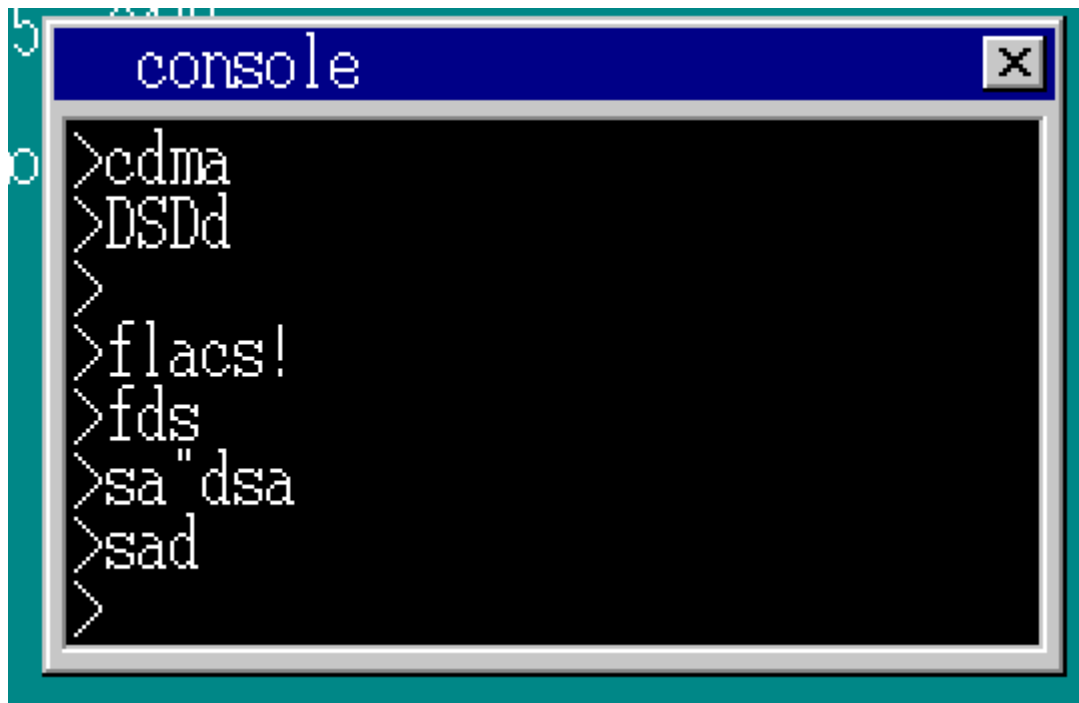
```
if (cursor_y < 28 + 112) {  
    putfonts8_asc_sht(sheet, cursor_x, cursor_y, COL8_FFFFFFFF, COL8_000000,  
        " ", 1);  
    cursor_y += 16;  
    putfonts8_asc_sht(sheet, 8, cursor_y, COL8_FFFFFFFF, COL8_000000, ">", 1);  
    cursor_x = 16;  
}
```



意料之内，打印到最后一行之后不再下滚了。

我们接下来要写向下滚动的逻辑。具体做法是将除第一行之外的每一行的东西都复制到上一行，然后我们用空行覆盖最后一行，就ok了。





工作正常

接下来我们要实现mem命令

mem: 在控制台中打印当前的内存使用情况

我们首先先去掉harimain当中在桌面的内存占用显示，把相关的代码注释掉就好了。

若要获取我们输入的内容，我们必须新开一个数组来记录我们都输入了什么东西。

```
if (cursor_x < 240) {
    s[0] = i - 256;
    s[1] = 0;
    cmdline[cursor_x / 8 - 2] = i - 256; // -2是因为有边框和提示符
    putfonts8_asc_sht(sheet, cursor_x, cursor_y, COL8_FFFFFFFF, COL8_000000, s,
        1);
    cursor_x += 8;
}
```

然后我们在处理回车的时候对cmdline进行一下判断。

在写这个之前，我们先把控制台换行单独拎出来写成一个函数。

```
int cons_newline(int cursor_y, struct SHEET *sheet)
{
    int x, y;
    if (cursor_y < 28 + 112) {
        cursor_y += 16;
    } else {
        for (y = 28; y < 28 + 112; y++) {
```

```

        for (x = 8; x < 8 + 240; x++) {
            sheet->buf[x + y * sheet->bysize] = sheet->buf[x + (y + 16) * sheet->bysize];
        }
    }
    for (y = 28 + 112; y < 28 + 128; y++) {
        for (x = 8; x < 8 + 240; x++) {
            sheet->buf[x + y * sheet->bysize] = COL8_000000;
        }
    }
    sheet_refresh(sheet, 8, 28, 8 + 240, 28 + 128);
}
return cursor_y;
}

```

然后修改task_cons中处理回车键的部分

```

cursor_y = console_newline(cursor_y, sheet); // 换行
if (strcmp(cmdline, "mem") == 0) { //使用strcmp来进行简化
    sprintf(s, "total %dMB", memtotal / (1024 * 1024));
    putfonts8_asc_sht(sheet, 8, cursor_y, COL8_FFFFFFFF, COL8_000000, s, 30);
    cursor_y = cons_newline(cursor_y, sheet);
    sprintf(s, "free %dKB", memman_total(memman) / 1024);
    putfonts8_asc_sht(sheet, 8, cursor_y, COL8_FFFFFFFF, COL8_000000, s, 30);
    cursor_y = cons_newline(cursor_y, sheet);
    cursor_y = cons_newline(cursor_y, sheet);
} else if (cmdline[0] != 0) {
    putfonts8_asc_sht(sheet, 8, cursor_y, COL8_FFFFFFFF, COL8_000000, "Command Illegal.",
sizeof("Command Illegal.));
    cursor_y = cons_newline(cursor_y, sheet);
    cursor_y = cons_newline(cursor_y, sheet);
}
putfonts8_asc_sht(sheet, 8, cursor_y, COL8_FFFFFFFF, COL8_000000, ">", 1); /// 打印提示符
cursor_x = 16;

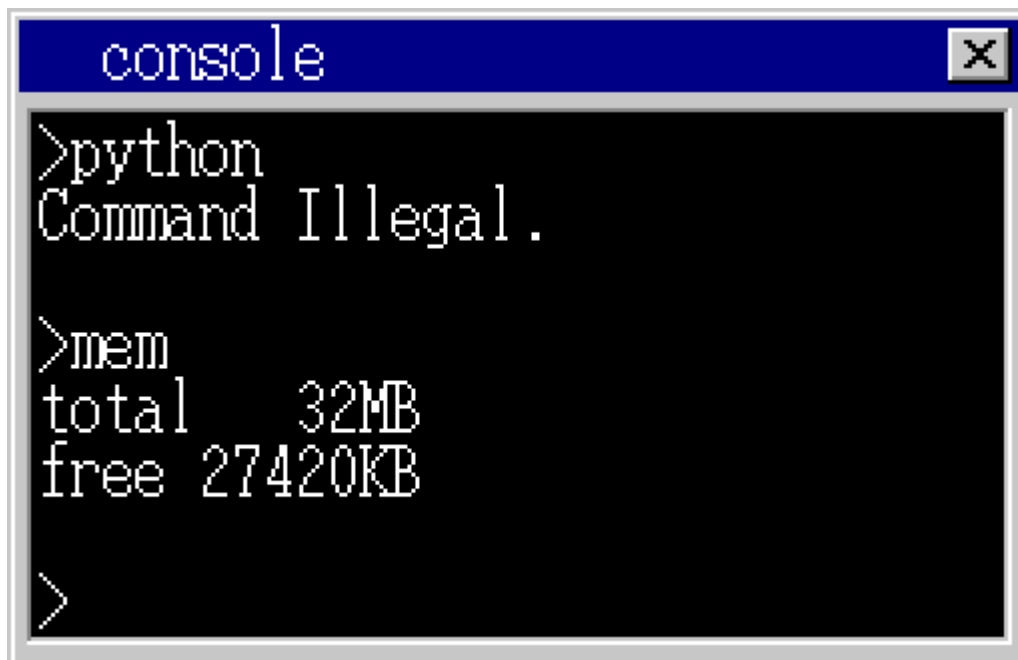
```

注意为了获取memtotal，我们使用了和获取sheet相同的trick

```

// void console_task(struct SHEET *sheet, unsigned int memtotal);
// ----- HariMain(void) -----
*((int *) (task_cons->tss.esp + 8)) = memtotal;

```

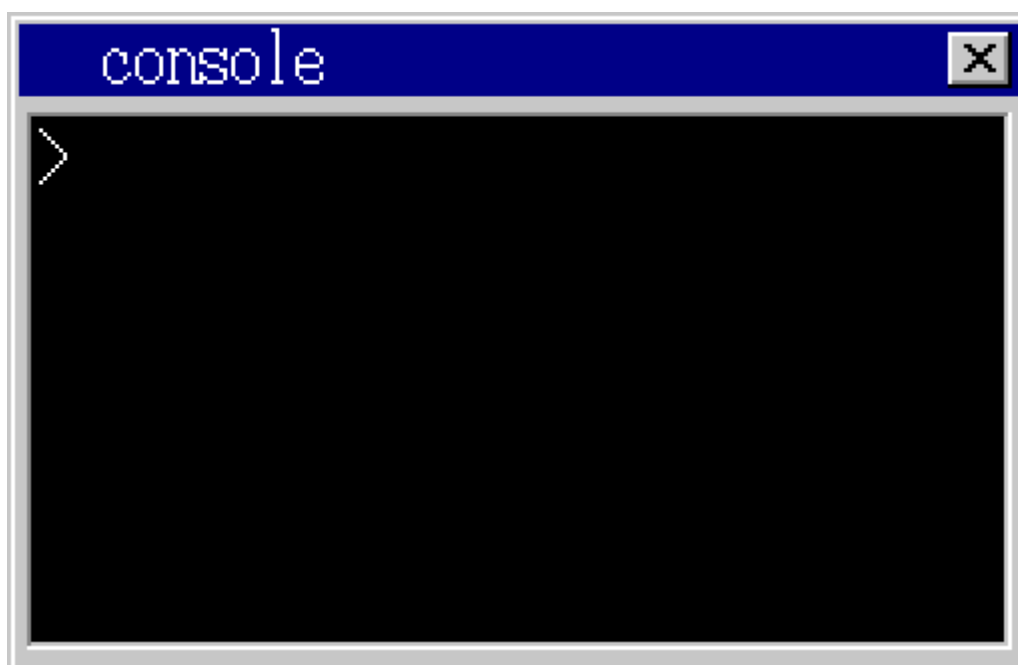


```
console
>python
Command Illegal.

>mem
total    32MB
free 27420KB

>
```

下面来实现cls命令，要点在于将cursor_x、cursor_y置为初始状态，把整个屏幕涂黑，然后打印提示符



```
console
>
```

最后来实现dir命令

dir命令是列举文件的命令

还记得文件名存储在磁盘的0x002600位置开始，也就是内存的0x00102600位置开始。

修改makefile，镜像中再添加几个文件

```

haribote.img : ip110.bin haribote.sys Makefile
$(EDIMG) imgin:../z_tools/fdimg0at.tek \
    wbinimg src:ip110.bin len:512 from:0 to:0 \
    copy from:haribote.sys to:@: \
    copy from:ip110.nas to:@: \
    copy from:make.bat to:@: \
    imgout:haribote.img

```

00002600	48 41 52 49 42 4f 54 45 53 59 53 20 00 00 00 00	H A R I B O T E S Y S
00002610	00 00 00 00 00 00 77 46 8a 4e 02 00 70 6c 00 00 w F 奇 . . p l . .
00002620	49 50 4c 31 30 20 20 20 4e 41 53 20 00 00 00 00	I P L 1 0 N A S
00002630	00 00 00 00 00 00 59 7a 42 35 39 00 95 0b 00 00 Y z B 5 9 . ? . .
00002640	4d 41 4b 45 20 20 20 20 42 41 54 20 00 00 00 00	M A K E B A T
00002650	00 00 00 00 00 00 f6 10 81 30 3f 00 2e 00 00 00 ? ? ?
00002660	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00002670	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

似乎有规律

结构是这样的

```

struct FILEINFO {
    unsigned char name[8], ext[3], type;
    char reserve[10];
    unsigned short time, date, clustno;
    unsigned int size;
};

```

开始的8个字节是文件名。文件名不足8个字节时，后面用空格补足。文件名超过8个字节的情况比较复杂，我们在这里先只考虑不超过8个字节的情况吧，一上来就挑战高难度的话，很容易产生挫败感呢。再仔细观察一下，我们发现所有的文件名都是大写的。如果文件名的第一个字节为0xe5，代表这个文件已经被删除了；文件名第一个字节为0x00，代表这一段不包含任何文件名信息。从磁盘映像的0x004200就开始存放文件haribote.sys了，因此文件信息最多可以存放224个。接下来3个字节是扩展名，和文件名一样，不足3个字节时用空格补足，如果文件没有扩展名，则这3个字节都用空格补足。扩展名和文件名一样，也全部使用了大写字母。后面1个字节存放文件的属性信息。我们这3个文件的属性都是0x20。一般的文件不是0x20就是0x00，至于其他的值，我们来看下面的说明。

0x02隐藏文件 0x04系统文件 0x08非文件信息（比如磁盘名称等） 0x10目录

接下来的10个字节为保留，也就是说，是为了将来可能会保存更多的文件信息而预留的，在我们的磁盘映像中都是0x00。话说，这个磁盘格式是由Windows的开发商微软公司定义的，因此，这段保留区域以后要如何使用，也是由微软公司来决定的。其他人要自行定义的话也可以，只不过将来可能会和Windows产生不兼容的问题。下面2个字节为WORD整数，存放文件的时间。因此即便文件的内容都一样，这里大家看到的数值也可能是因人而异的。再下面2个字节存放文件的日期。这些数值虽然怎么看都不像是时间和日期，但只要用微软公司的公式计算一下，就可以转换为时、分、秒等信息了。接下来的2个字节也是WORD整数，代表这个文件的内容从磁盘上的哪个扇区开始存放。变量名clustno本来是“簇号”（cluster number）的缩写，“簇”这个词是微软的专有名词，在这里我们先暂且理解为和“扇区”是一码事就好了。最后的4个字节为DWORD整数，存放文件的大小。

以上引用的是书上的内容。

不过我比较好奇，好多文件的扩展名不止3个字节，不知道这是怎么处理的

于是我稍微修改了下makefile


```
haribote.img : ipl10.bin haribote.sys Makefile
$(EDIMG) imgin:../z_tools/fdimg0at.tek \
  wbinimg src:ipl10.bin len:512 from:0 to:0 \
  copy from:haribote.sys to:@: \
  copy from:ipl10.nass to:@: \
  copy from:make.bat to:@: \
  imgout:haribote.img
```

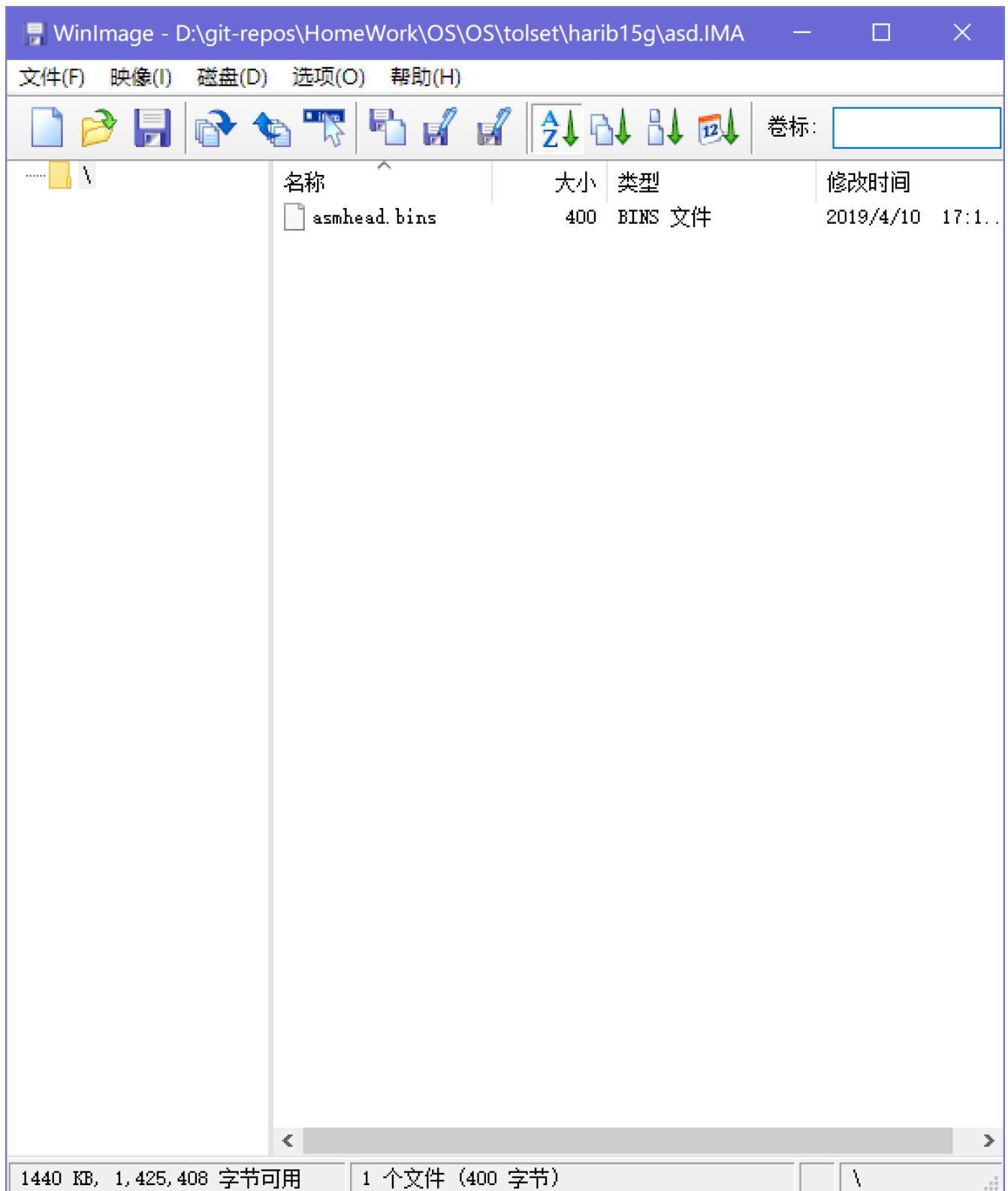
并且把ipl.nas重命名成了ipl.nass, 重新make并且用二进制编辑器查看

00002600	48	41	52	49	42	4f	54	45	53	59	53	20	00	00	00	00
00002610	00	00	00	00	00	00	aa	49	8a	4e	02	00	70	6c	00	00
0000262a	49	50	4c	31	30	20	20	20	4e	41	53	20	00	00	00	00
00002630	00	00	00	00	00	00	a2	49	8a	4e	39	00	95	0b	00	00
00002640	4d	41	4b	45	20	20	20	20	42	41	54	20	00	00	00	00
00002650	00	00	00	00	00	00	f6	10	81	30	3f	00	2e	00	00	00

H A R I B O T E S Y S															
. 狗 箭 . . p l . .															
I P L 1 0 N A S															
. 箭 9 . ? . .															
M A K E B A T															
. ? ? ?															

emmm, 只进行了简单的截断? 我觉得是edimg的锅。

于是我下载了winimg, 并尝试进行修改



00002600	e5 53 4d 48 45 41 44 20 42 49 4e 00 18 67 6c 86	鎔 MHEAD BIN..gl 噢
00002610	8a 4e 00 00 63 03 a5 89 8a 4e 40 00 90 01 00 00	N..c. 裔 @.? ..
00002620	41 61 00 73 00 6d 00 68 00 65 00 0f 00 e6 61 00	Aa.s.m.h.e...鎔 .
00002630	64 00 2e 00 62 00 69 00 6e 00 00 00 73 00 00 00	d...b.i.n...s...
00002640	41 53 4d 48 45 41 7e 31 42 49 4e 00 18 67 6c 86	ASMHEA~1BIN..gl 噢
00002650	8a 4e 00 00 63 03 a5 89 8a 4e 40 00 90 01 00 00	N..c. 裔 @.? ..
00002660	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

似乎是ok的，关闭重新打开也可以正常读取。不过格式看不太出来，于是网搜

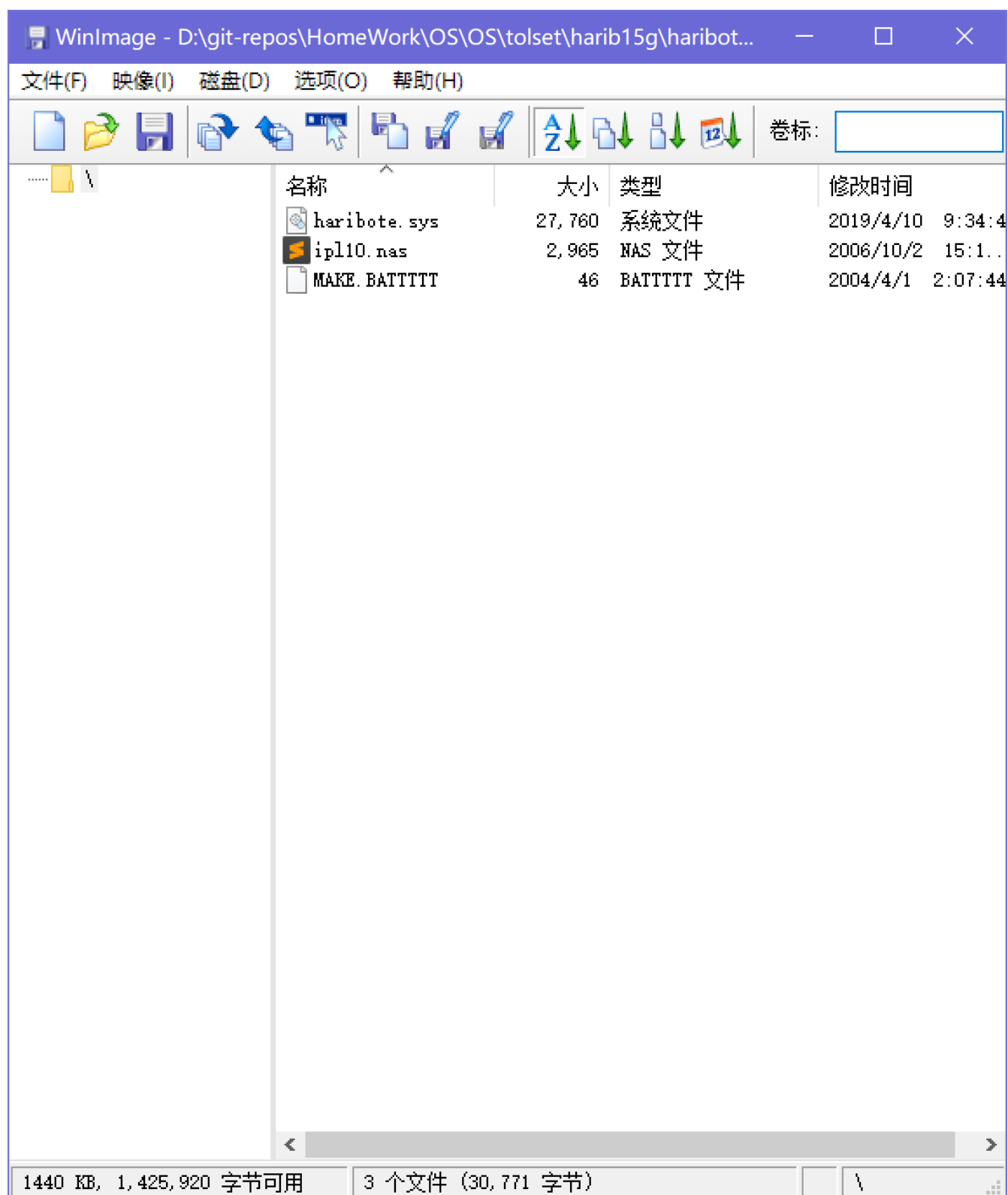
<https://blog.csdn.net/sikuon/article/details/76397831>

emmm, 看起来不ok? 不知道这怎么实现的, 暂时留作问题

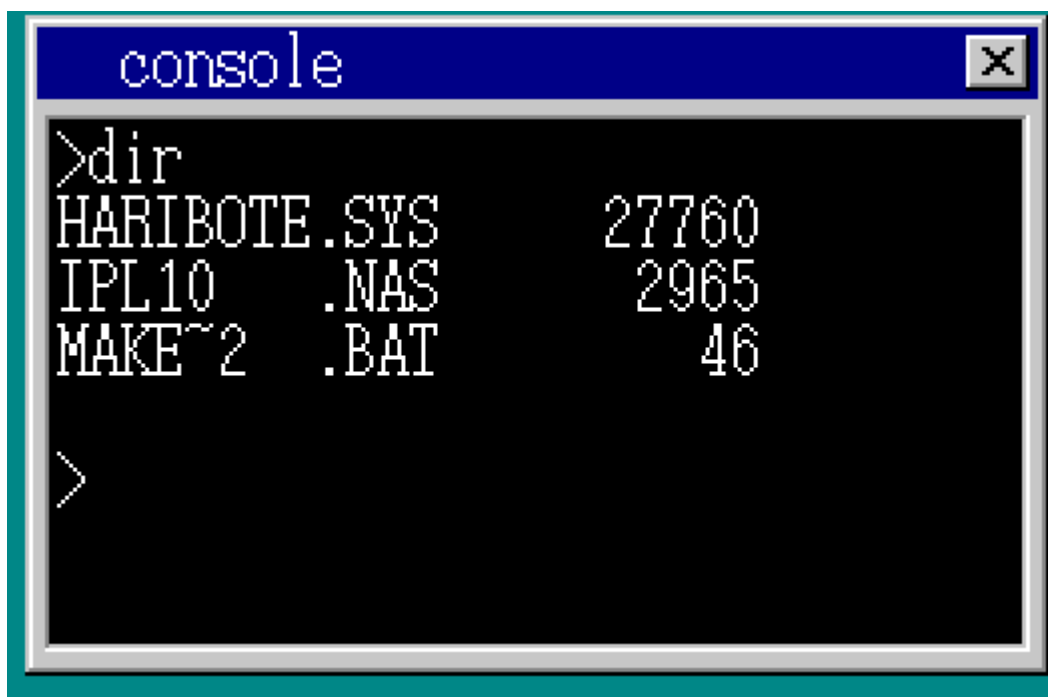
然后我们就可以来编制dir了!

```
for (x = 0; x < 224; x++) {
    if (finfo[x].name[0] == 0x00) {
        break;
    }
    if (finfo[x].name[0] != 0xe5) {
        if ((finfo[x].type & 0x18) == 0) {
            sprintf(s, "filename.ext %7d", finfo[x].size);
            for (y = 0; y < 8; y++) {
                s[y] = finfo[x].name[y];
            }
            s[ 9] = finfo[x].ext[0];
            s[10] = finfo[x].ext[1];
            s[11] = finfo[x].ext[2];
            putfonts8_asc_sht(sheet, 8, cursor_y, COL8_FFFFFFFF,
                               COL8_000000, s, 30);
            cursor_y = cons_newline(cursor_y, sheet);
        }
    }
}
```

我make之后又皮了一下, 似乎有些发现



这样改完之后



后面有个~2，又修改为make.batt，变成~1，查看二进制。

他似乎是用~作为转义符，之前的两行存储了更多的关于文件名的更多信息。

00002660	41	4d	00	41	00	4b	00	45	00	2e	00	0f	00	5f	42	00
00002670	41	00	54	00	54	00	54	00	54	00	00	00	54	00	00	00
00002680	41	41	41	45	00	00	00	40	41	54	00	00	00	00	00	00

A	M	.	A	.	K	.	E	_	B	.
A	.	T	.	T	.	T	.	T	.	T	.	T	.	T	.	T	.	.
M	A	K	E

今天就先到这吧