在 linux 上模拟 uCOS-II 实时操作系统

1 引言

uCOS-II 是一个短小而功能强大的实时嵌入式操作系统。在 Jean J. Labrosse 先生所着. 由邵贝贝教授翻译的 Micro/uCOS-II THE REAL-TIME KERNEL (Second Edition)一书中对这一操作系统作出了精彩的讲解，该书是一部关于 uCOS-II 操作系统的经典教材，同时在书中提供了关于 uCOS-II 在 windows 环境下的移植的 4 个范例。本文对其第一个范例作一个在 redhat linux9.0 上的移植版本。移植的工作主要集中在三个方面，下文将分为三章，结合代码详细介绍。文章的最后将介绍我的试验平台并演示我的试验结果。

2 字符串的显示

字符串显示函数 PC-DispStr 在文件 pc.c 中，这个文件本身不是 uCOS-II 的一部分。它的主要工作是建立一系列的功能函数来发挥 PC 机的强大功能，并被测试代码所调用。

2.1 设置前景色和后景色

我们使用类似于 prinf("\033[30m")的语句来设置颜色. 转义序列就是一个让 shell 执行一个特殊步骤的控制指令。转义序列通常都是以 ESC 开头(这也是它的命名原因)。在 shell 里表示为 [。这种表示法需要一点时间去适应， 也可以用\033 完成相同的工作(ESC 的 ASCII 码用十进制表示就是 27，=用八进制表示的 33)。\033 声明了转义序列的开始，然后是[开始定义颜色。下面我们要选择前景色(这里是 32，代表绿色)。背景色的 40 表示黑色。要是不想让提示符后面的文字也变成绿色，我们用\033[0m 关闭转义序列，\033[0m 是 shell 的默认颜色。前景色和背景色都有 8 种可用的选择。可选颜色：红色、绿色、黄色、蓝色、洋红、青色和白色。他们对应的颜色代码是：30(黑色)、31(红色)、32(绿色)、33(黄色)、34(蓝色)、35(洋红)、36(青色)、37(白色)。用同样色方法设置背景色，不过要把第一个数字"3"替换成"4"，例如 40、41、42、43、44、45、46、47。虽然在这里可以按照上面介绍的对应关系定义修改在 pc.h 中定义的前景色和后景色的宏，使对应关系更加明确。(注意：他的后面一位表示前景色，前面一位表示后景色)，但是我们在这里的设计思路是尽量不对原书中的代码作改动，所以在函数的实现中直接使用 switch 语句，对相应的前景色和后景色 。(linux 的 shell 只支持以上几种颜色)

```
switch (color&0xF0) /*查看前景色*/
{ case DISP_FGND_BLACK: printf("\033[30m");break;
……
}
```

```
switch(color&0x0F) /*查看后景色*/

{ case DISP_BGND_BLACK: prinf ("\033[40m");

break;

……

}
```

2.2 跟踪光标的位置

我使用 printf("\033[%u;%uH",y+1,x+1)来跟踪光标的位置。33 是声明了转义序列的开始，上文已经介绍，不再累叙，[y;xH 是设置光标位置的格式。x 和 y 分别表示横轴和纵轴。

3 键盘输入

键盘输入函数 PC_GetKey 在 windows 环境下，由于有库函数 kbhit 返回最近所敲的按键.就很容易实现。而在 linux 环境下我们需要构造自己的 kbhit，在参考文献 2 中 John.Wiley.Sons 先生提供了一种现成的实现方法(这个方法会阻塞 read 函数.在本文中并不适用)，这里我们使用了另外的一种实现方法，下面介绍给出其实现代码.

```
int kbhit(void){

struct timeval tv;

fd_set readFd;

struct termios newKbdMode;

if(!inited){

newKbdMode.c_lflag&=~(ICANON | ECHO);

newKbdMode.c_cc[VTIME]=0;

newKbdMode.c_cc[VMIN]=1;

tcsetattr(0,TCSANOW,&newKbdMode);

atexit(rekbd);

inited=1;
```

```
}

tv.tv_sec=0;

tv.tv_usec=0;

FD_ZERO(&readFd);

FD_SET(STDIN_FILENO,&readFd);

select(1,&readFd,NULL,NULL,&tv);

if(FD-ISSET(STDIN-FILENO, &readFd))

return 1;

else

return 0;

}
```

3.1 控制台的初始化

首先，这里使用了全局变量 inited，它是一个初始化与否的标记.因为函数 kbhit 将被多次调用，而初始化只需要做一次.这样.当发现 inited 置 1 以后，就不会去做重复性的初始化工作了。如果 inited 为 0，就需要对控制台(键盘)做初始化工作，这里定义了内核结构体 termios 类型的变量 newKbdMode，我们需要对这个结构体的两个成员 c_lflag 和 c_cc 进行初始化，代码中对 c_lflag 的设置表示终端为不回显的非标准模式。c_cc[VTIME]=0，c_cc[VMIN]=1 表示读函数会等待.直到出现 1 个键盘输入为止。(关于这个结构体的详细分析，可参阅参考文献 2 的第 5 章)。然后再调用 tcsetattr 把设置的值写入。最后，函数 atexit 将在 3.3 节详叙。

3.2 检测键盘的输入

在这里我们使用宏 FD_ZERO 把内核的结构体 readFd 清 0.用宏 FD_SET 把标准输入的文件描述符 STDIN_FILENO 和 readFd 关联，然后用 select 函数来监测输入.他只关注一个描述符，所以第一个参数为 1，第二个参数为上面的 readFd，后面的两个参数表示是否关注标准输出和出错的文件描述符，我们不要，所以置 0.最后一个参数表示超时时间，我们不需要，所以置 0。经过以上的处理后，如果有输入时.宏 FD_ISSET 就会返回非 0 值。我们就知道键盘上有输入。

3.3 系统退出

在windows环境下使用了成对的函数PC_DOSSaveReturn()和PC_DOSReturn。前一个保存DOS的状态，后一个在退出时前调用.恢复保存的DOS状态。而在linux下，表面看来我仅使用函数exit()直接退出，而没有进行类似的保存一恢复处理.但实际上在linux下我们调用了函数atexit(function)来设置程序正常结束前调用的函数，当程序通过调用exit()返回时，参数function所指定的函数会先被调用.然后才真正由exit()结束程序。function将指定函数rekbd(函数的实现见下面的代码)，这个函数就是清屏和清处所有前文的属性设置，\033声明了转义序列的开始，然后是[2J，表示清屏。[0m表示关闭所有属性。

```
void rekbd(void){

prinf("\033[0m");

prinf("\033[2J");

}
```

4 MAKEFILE 文件的编写

在Jean J.Labrosse先生的原书中是使用boland c的编译器.而我们在linux下使用GCC的编译器，由于编译器的改变.所以makefile就需要重写。为了简化makefile的编写，我提供一种最简单的方法，那就是把所有uCOS-II的源码(SOFTWAREuCOS-IISOURCE). 以及配置头文件和测试函数(SOFTWAREuCOS-IIEX1_x86LBC45SOURCE).还有按上文编写的pc.c和pc.h文件，全部放在linux的根目录下.假设为/test78，则makefile可简写为如下方式：

```
UCOS_SRC=/test78

UCOS_PORT=/test78

UCOS_PC=/test78

all:

gcc-I$(UCOS_SRC) -I$(UCOS_PORT) -I$(UCOS_PC) test.c
$(UCOS_SRC)/uCOS_II.C $(UCOS_PC)/

pc.c $(UCOS_PORT)/os_cpu_c.c -o test
```

all是一个伪目标，"伪目标"并不是一个文件，只是一个标签，它的特性是，总是被执行的。这样的目的是让编译器每次都产生新的目标。-o test指定输出文件为test.'-I'选项指定搜索的目录.

注意：把所有源文件都放在一个目录下也许并不是一个好方法，它使得整个工程杂乱无章，特别是在工程比较大时.是不能这样处理的。但这里仅仅是为了

简化 makefile 的编写，提供一个可行的方法。所以在这个 makefile 的前面，我定义了几个宏，如果需要编译的几个文件在路径下，就只需要指定路径就可以了。

5 结束语

本文的创新点主要体现在

1. 自建的键盘输入函数。由于(Beginning.Linux.Programming)中实现会阻塞 read 函数，所以本文采用了改进的方法实现键盘输入，详见第 3 节。

2. MAKEFILE 文件。由于编译器的改变，我们需要改写 makefile 文件，本文提供了一种非常简单的编写方法，详见第 4 节。

我的试验平台如下：在 Virtual PC 2004 上安装 red hat linux 9.0，并且在 linux 下进行编译和调试。