

## 4 实验三 VGA 与串行端口

### 4.1 实验目的

1. 了解 Linux 中 VGA 的实现，掌握 `printk` 等字符打印函数的实现方法。
2. 了解 Linux 通过串行端口与终端交换信息的函数实现。

**视频图形阵列** (Video Graphics Array, 简称 VGA) 是 IBM 于 1987 年提出的一种电脑显示标准。运用该标准的接口被称为 VGA 端子，通常用于在电脑的显示卡、显示器及其他设备发送模拟信号。本次实验关注于内核对彩色字符模式显示缓冲区的控制。

### 4.2 80 \* 25 彩色字符模式显示缓冲区的结构

内存地址中，B8000h \ BFFFFh 共 32KB 的空间，是 80\*25 彩色字符模式的显示缓冲区。在这个地址空间中写入数据，写入的内容会立即出现在显示器上。

在 80\*25 彩色字符模式下，显示器可显示 25 行，每行 80 个字符，每个字符可以有 256 种属性，包括背景色、前景色（即字体色）、闪烁、高亮等组合而成的属性。

一个字符在显示缓冲区中占用 2 个字节，分别存放其 ASCII 码和属性。一屏的内容在显示缓冲区中共占用占  $2 * 25 * 80 = 4000$  字节。

显示缓冲区分 8 页，每页 4 KB，显示器可显示任意一页的内容。一般地，显示第 0 页内容，即显示 B8000H \ B8F9FH 中的共 4000 字节的内容。

在一页显示缓冲区中，一行的 80 个字符占用 160 字节，偏移 000 \ 09f 对应显示器上的第一行，偏移 0A0 \ 13f 对应显示器上的第二行，以此类推。

在属性字节中，闪烁、背景色、高亮与前景色是按位设置的，如表 4.

下面介绍内核中用于 VGA 的一些函数。

#### 4.2.1 函数 `video_putchar_at(char ch, int x, int y, char attr)`

本函数用于在屏幕指定位置 (x,y) 处输出指定字符串 `ch`，并且指定字符串 `ch` 后光标的颜色 `attr`。其中，`x` 是行数，`y` 是列数。

表 4: 属性字节中每位的具体含义

7	6	5	4	3	2	1	0
BL	R	G	B	I	R	G	B
闪烁	背景色	背景色	背景色	高亮	前景色	前景色	前景色

ch 与 attr 变量不需要额外的处理，仅需要将这两个变量赋值到合适的内存地址处。

#### 4.2.2 不定参数函数

在内核调用函数 printk 时，需要被格式化输出的内容长度是不定的，因此 printk 是一个不定参数函数。函数 printk 的实现中，将用到 <stdarg.h> 库中的数据结构或函数：va\_list、va\_start、va\_arg。<sup>13</sup>

#### 4.2.3 函数 printk(char \*fmt, ...)

printf 与 printk 在实现的功能上几乎一致，根据使用 C 语言函数 printf 的经验，可知这个格式化输出函数的首个参数 char \*fmt 应当表示一段字符串的首位，以 \*fmt 为首的字符串描述了格式化输出的格式。

这个字符串中可能含有以 “%” 为首位的子串，表示一个数字、一个 char 类字符或一段字符串，它具体的值即 printk 的某个在 \*fmt 之后的参数。

**练习** 打开 exp3/kernel/printk.c，尝试依次实现 video\_putchar\_at 和 printk 函数。

首先，思考一个屏幕坐标 (x, y) 对应的显存地址是如何计算的？其中，显存地址的首位在 exp3/kernel/printk.c 中由 video\_buffer 指针表示。

请通过调用函数 printnum、video\_putchar、va\_start、va\_arg，尝试为函数 printk 实现如下功能<sup>a</sup>：

<sup>13</sup><https://msdn.microsoft.com/zh-cn/library/kb57fad8.aspx>

1. 字符串无 “%” 时能直接在屏幕上输出字符串的内容
2. 读取到 “%d”，根据某个参数输出一个有符号十进制整数
3. 读取到 “%u”，根据某个参数输出一个无符号十进制整数
4. 读取到 “%x”，根据某个参数输出一个无符号十六进制整数
5. 读取到 “%c”，根据某个参数输出一个字符
6. 读取到 “%s”，根据某个参数输出一段字符串
7. 读取到 “%%”，输出百分号 “%”

注意，你需要仔细考虑各种状态转移。例如，你可能认为读取到 “%” 时是解析一项参数的开始，但它同样可能是该字符串的最后一个字符，即一个百分号可能没有后继的字符。

若函数 `video_putchar_at` 与函数 `printk` 已成功实现，使用 QEMU 运行 `neuos`，将见到如图 8 的界面。`printk` 函数是由 `exp3/kernel/main.c` 调用的。

<sup>a</sup>参考网站：[https://wiki.osdev.org/Printing\\_To\\_Screen](https://wiki.osdev.org/Printing_To_Screen)



图 8: 函数 `printk` 成功实现

**练习** 尝试通过调用 `printk.c` 中的两个函数 `memcpy` 与 `video_putchar_at` 实现滚屏函数 `roll_screen`. 滚屏函数被调用的条件应当是，需要输出字符到当前屏幕的最后一行之后。因此需要将当前屏幕的内容整体上移，使得屏幕最后一行能够输出新的内容。

要验证函数 `roll_screen` 是否成功实现，可打开目录 `exp3/kernel/main.c`,

使用函数 `printk` 输出足够多的字符，再运行 `neuos`，测试能否成功滚屏。

### 4.3 串行端口

**串行端口** (Serial port) 主要用于串行式逐位数据传输。可用于连接外置调制解调器、打印机、路由器等设备。在消费电子领域已被 USB 替代，在网络设备中仍是主要的传输控制方式。

通过串行接口，Linux 设备，即使并非计算机，也可实现与其他设备的通信；本实验中的串行端口用于将 `neuos` 中的信息打印到 NEU-OS Lab Environment 的终端中。

**练习** 进入 `exp3/kernel` 目录，打开 `serial.c` 源文件。

函数 `is_transit_empty` 用于判断是否能够传输，若允许传输，函数返回 0。

结合函数 `is_transit_empty` 与端口输出函数 `outb`，实现函数 `s_putchar`。参考 Serial Ports - OSDev Wiki 4.3 节<sup>a</sup>。

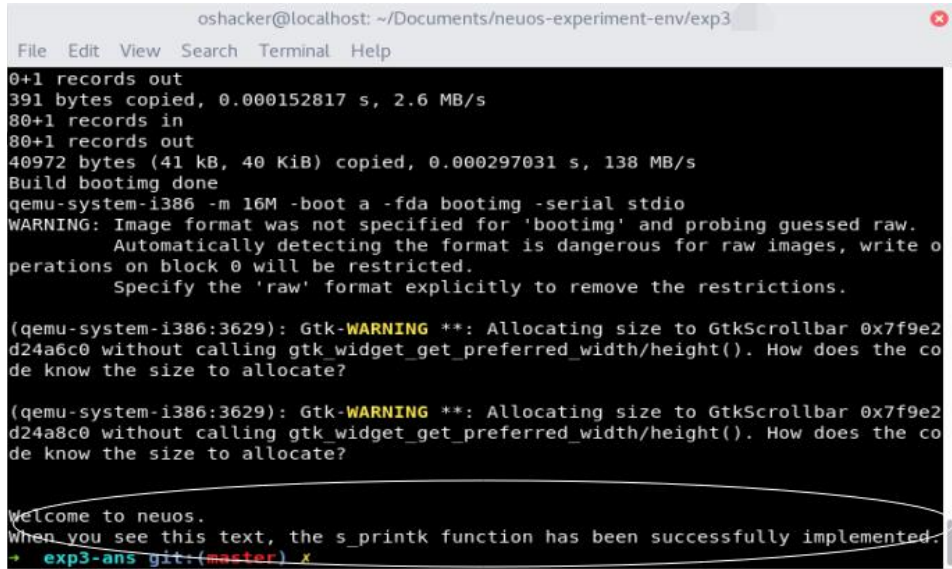
然后，类似于 `printk` 函数的实现，请通过调用函数 `s_printnum`、`s_putchar`、`va_start`、`va_arg`，尝试为函数 `s_printk` 实现如下功能：

1. 字符串无 “%” 时能直接在屏幕上输出字符串的内容
2. 读取到 “%d”，根据某个参数输出一个有符号十进制整数
3. 读取到 “%u”，根据某个参数输出一个无符号十进制整数
4. 读取到 “%x”，根据某个参数输出一个无符号十六进制整数
5. 读取到 “%c”，根据某个参数输出一个字符
6. 读取到 “%s”，根据某个参数输出一段字符串
7. 读取到 “%%”，输出百分号 “%”

若函数 `s_putchar` 与函数 `s_printk` 已成功实现，使用 QEMU 运行 `neuos`，运行结果如图 9 所示，字符被输出到 `os` 的终端而不是 QEMU

界面中。

<sup>a</sup>[http://wiki.osdev.org/Serial\\_Ports](http://wiki.osdev.org/Serial_Ports)



```
oshacker@localhost: ~/Documents/neuos-experiment-env/exp3
File Edit View Search Terminal Help
0+1 records out
391 bytes copied, 0.000152817 s, 2.6 MB/s
80+1 records in
80+1 records out
40972 bytes (41 kB, 40 KiB) copied, 0.000297031 s, 138 MB/s
Build bootimg done
qemu-system-i386 -m 16M -boot a -fda bootimg -serial stdio
WARNING: Image format was not specified for 'bootimg' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

(qemu-system-i386:3629): Gtk-WARNING **: Allocating size to GtkScrollbar 0x7f9e2
d24a6c0 without calling gtk_widget_get_preferred_width/height(). How does the co
de know the size to allocate?

(qemu-system-i386:3629): Gtk-WARNING **: Allocating size to GtkScrollbar 0x7f9e2
d24a8c0 without calling gtk_widget_get_preferred_width/height(). How does the co
de know the size to allocate?

Welcome to neuos.
When you see this text, the s_printk function has been successfully implemented.
+ exp3-ans git:(master) x
```

图 9: 函数 s\_printk 成功实现

**练习** 在 exp3/kernel/serial.c 源文件的末尾, 尝试实现通过串行端口读取信息的代码。仅尝试完成代码<sup>a</sup>, 不必实现最终效果。

<sup>a</sup>参考 [http://wiki.osdev.org/Serial\\_Ports](http://wiki.osdev.org/Serial_Ports) 读取数据部分。

**拓展学习** 找到 printk.c 源文件 208 行以后的“拓展学习”部分, 实现具有指定 8 种颜色功能的函数 echo; echo 需要调用 video\_putchar\_color 等函数。

实现效果参考 [http://misc.flogisoft.com/bash/tip\\_colors\\_and\\_formatting](http://misc.flogisoft.com/bash/tip_colors_and_formatting), 考虑到实现的难度等因素, 以下描述与该参考网站的描述并不完全一致。

例如, echo(“\033[x;ymneuos”); 可将“neuos”以 x 为前景色, y

表 5: 前景色与背景色代码表示的颜色

x: 字体色	y: 背景色	颜色	RGB 二进制
0	0	BLACK	000
1	1	RED	100
2	2	GREEN	010
3	3	YELLOW	110
4	4	BLUE	001
5	5	MAGENTA	101
6	6	CYAN	011
7	7	WHITE	111

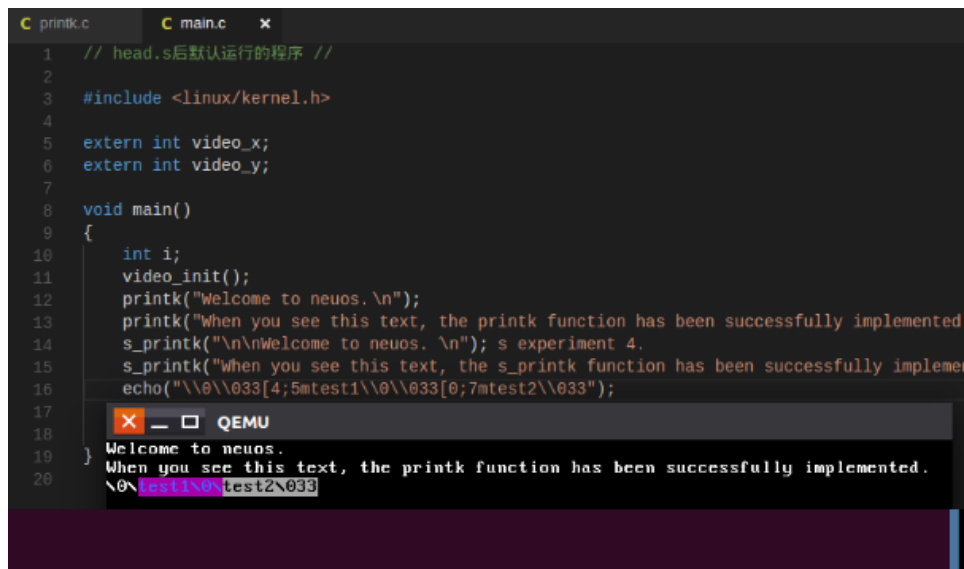
为背景色输出。其中 x 和 y 是两个数字，均是 0~7 的整数，其表示的颜色见表 5。m 作为颜色的唯一后缀。

如果没有 “\033[” 指定颜色，函数 echo 应直接输出文本。这个格式是严格的，不符合这个格式的字符串应按普通文本直接输出。函数应支持输出多个指定颜色的字符串。

表 5 中的 RGB 指的是表 4 中属性字节的第 0~2 位及第 4~6 位；为了更好的显示效果，此处指定属性字节的**第 3 位置为 1，第 7 位置为 0**。根据这些信息，并结合表 4，可计算出应填充到属性字节的十六进制数。

如要测试该函数是否正确实现，在 exp3/kernel/main.c 中调用函数 echo 并填入参数即可。实现效果如图 10 所示。

注意，C 语言中 ‘\\’ 是斜杠的转义字符，‘\0’ 是 NULL 的转义字符。



The image shows a QEMU virtual machine window with two tabs: 'C printk.c' and 'C main.c'. The 'C main.c' tab is active, displaying the following C code:

```
1 // head.s后默认运行的程序 //
2
3 #include <linux/kernel.h>
4
5 extern int video_x;
6 extern int video_y;
7
8 void main()
9 {
10     int i;
11     video_init();
12     printk("Welcome to neuos.\n");
13     printk("When you see this text, the printk function has been successfully implemented\n\nWelcome to neuos. \n");
14     s_printk("When you see this text, the s_printk function has been successfully implemented\n\nWelcome to neuos. \n");
15     s_printk("When you see this text, the s_printk function has been successfully implemented\n\nWelcome to neuos. \n");
16     echo("\\0\\033[4;5mtest1\\0\\033[0;7mtest2\\033");
17 }
18
19 }
20
```

Below the code editor, the QEMU window title bar shows 'QEMU'. The output of the program is displayed in the console area:

```
Welcome to neuos.
When you see this text, the printk function has been successfully implemented.
\\0\\033[4;5mtest1\\0\\033[0;7mtest2\\033
```

图 10: 函数 echo 实现效果