LISS
信息安全研究所

操作系统设计及实践

《操作系统原理》配套实验

信安系操作系统课程组

2024年12月

操作系统设计实验系列（九）

I/O子系统

武漢大学

一、实验目标

1. 了解键盘输入的基本原理。
2. 了解显示器输出的基本原理。
3. 了解 TTY 终端的概念。
4. 了解 I/O 相关系统调用的实现及其对应功能。

二、本次实验基本内容

1. 验证键盘扫描码与键盘中断的过程。
2. 分析显示器的显示操控，以及输出方式。
3. 验证分析 tty 的处理过程，以及如何支持进程对 tty 的读写。
4. 结合 printf 的实现代码，理解 I/O 系统调用的实现机理

三、本次实验要解决的问题

1. 验证键盘输入的处理程序，并回答如下问题：
   1. 解释Scancode、MakeCode、BreakCode的区别
   2. 解释本实验如何解析扫描码
   3. 解释键盘输入缓冲区的作用，以及处理过程
   4. 为什么在/c/的kliba.asm，代码7.12中，需要disable_int, enable_int?
2. 解释一下，重新设置显示开始地址的原理。
3. 解释多个tty，能够并发处理输入输出的原理，结合代码给出一个比书上框图更为细致的流程图。
4. 动手做1：请添加一个你自己个性化的TTY，在这个TTY上，你可以根据键盘输入或者移动光标的某种规律，运行一个你的彩蛋程序，而在其他TTY中不会有这个效果。
5. 动手做2：尝试扩展一下printf，让它支持%s，想想目前的printf实现是否有什么安全漏洞？可以怎么解决。

四、需要了解的一些知识

1. 键盘的处理基本原理

- 物理连接方式
  - 键盘编码器
  - 键盘控制器
  - 中断控制器
- 键盘的敲击过程
  - 按下状态
  - 保持按住状态
  - 放开状态
- 扫描码
  - Make Code
  - Break Code

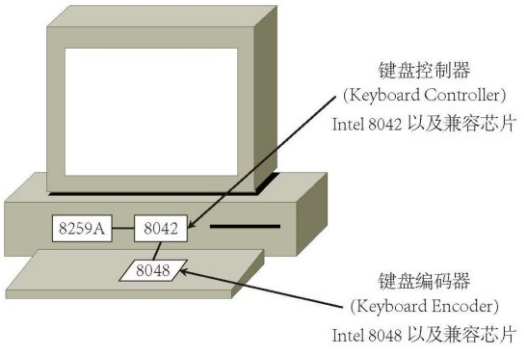


表7.2 Scan code set1扫描码

Key	Make	Break	Key	Make	Break
A	1E	9E	Tab	0F	8F
B	30	B0	C Lock	3A	BA
C	2E	AE	L Shift	2A	AA
D	20	A0	L Ctrl	1D	9D
E	12	92	L GUI	E0,5B	E0,DB
F	21	A1	L Alt	38	B8
G	22	A2	R Shift	36	B6
H	23	A3	R Ctrl	E0,1D	E0,9D
I	17	97	R GUI	E0,5C	E0,DC
J	24	A4	R Alt	E0,38	E0,B8
K	25	A5	APPS	E0,5D	E0,DD
L	26	A6	Enter	1C	9C
M	32	B2	Esc	1	81
N	31	B1	F1	3B	BB
O	18	98	F2	3C	BC
P	19	99	F3	3D	BD
Q	10	19	F4	3E	BE
R	13	93	F5	3F	BF
S	1F	9F	F6	40	C0
T	14	94	F7	41	C1
U	16	96	F8	42	C2
V	2F	AF	F9	43	C3
W	11	91	F10	44	C4
X	2D	AD	F11	57	D7
Y	15	95	F12	58	D8
Z	2C	AC	Print Screen	E0,2A E0,37	E0,B7 E0,AA

（续表）

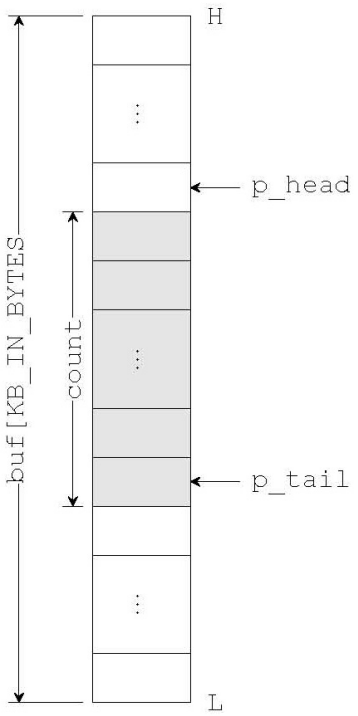
Key	Make	Break	Key	Make	Break
0	0B	8B	Scroll	46	C6
1	2	82	Pause	E1,1D 45,E1 9D,C5	NONE
			[	1A	9A
			Insert	E0,52	E0,D2
			Home	E0,47	E0,C7
			PgUp	E0,49	E0,C9
			Delete	E0,53	E0,D3
			End	E0,4F	E0,CF
			PgDn	E0,51	E0,D1
			↑	E0,48	E0,C8
			←	E0,4B	E0,CB
			↓	E0,50	E0,D0
			→	E0,4D	E0,CD
			Num	45	C5
			Backspace	0E	8E
		B5	KP *	37	B7
			KP +	4E	CE
			KP EN	E0,1C	E0,9C
			Next Track*	E0,19	E0,99
			Previous Track*	E0,10	E0,90
			Stop*	E0,24	E0,A4
			Play/Pause*	E0,22	E0,A2
			Mute*	E0,20	E0,A0
			Volume Up*	E0,30	E0,B0
			Volume Down*	E0,2E	E0,AE
			Media Select*	E0,6D	E0,ED
			E-mail*	E0,6C	E0,EC
			Calculator*	E0,21	E0,A1
			My Computer*	E0,6B	E0,EB
			WWW Search*	E0,65	E0,E5
			WWW Home*	E0,32	E0,B2
			WWW Back*	E0,6A	E0,EA
.	34	B4	WWW Forward*	E0,69	E0,E9
/	35	B5	WWW Stop*	E0,68	E0,E8
Power	E0,5E	E0,DE	WWW Refresh*	E0,67	E0,E7
Sleep	E0,5F	E0,DF	WWW Favorites*	E0,66	E0,E6
Wake	E0,63	E0,E3			

四、需要了解的一些知识

1. 键盘的处理基本原理
   - 扫描码的解析

代码7.5 扫描码解析数组（chapter7/c/include/keymap.h）

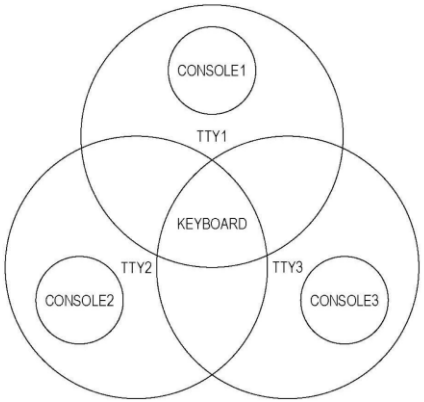
```
19  u32 keymap[NR_SCAN_CODES * MAP_COLS] = {
20
21  /* scan-code                    !Shift        Shift        E0 XX */
22  /* ============================================================== */
23  /* 0x00 - none          */     0,            0,           0,
24  /* 0x01 - ESC           */     ESC,          ESC,         0,
25  /* 0x02 - '1'           */     '1',          '!',         0,
26  /* 0x03 - '2'           */     '2',          '@',         0,
27  /* 0x04 - '3'           */     '3',          '#',         0,
28  /* 0x05 - '4'           */     '4',          '$',         0,
29  /* 0x06 - '5'           */     '5',          '%',         0,
30  /* 0x07 - '6'           */     '6',          '^',         0,
31  /* 0x08 - '7'           */     '7',          '&',         0,
32  /* 0x09 - '8'           */     '8',          '*',         0,
33  /* 0x0A - '9'           */     '9',          '(',         0,
34  /* 0x0B - '0'           */     '0',          ')',         0,
35  /* 0x0C - '-'           */     '-',          '_',         0,
36  /* 0x0D - '='           */     '=',          '+',         0,
37  /* 0x0E - BS            */     BACKSPACE,    BACKSPACE,   0,
38  /* 0x0F - TAB           */     TAB,          TAB,         0,
39  /* 0x10 - 'q'           */     'q',          'Q',         0,
40  /* 0x11 - 'w'           */     'w',          'W',         0,
41  /* 0x12 - 'e'           */     'e',          'E',         0,
42  /* 0x13 - 'r'           */     'r',          'R',         0,
43  /* 0x14 - 't'           */     't',          'T',         0,
44  /* 0x15 - 'y'           */     'y',          'Y',         0,
45  /* 0x16 - 'u'           */     'u',          'U',         0,
46  /* 0x17 - 'i'           */     'i',          'I',         0,
47  /* 0x18 - 'o'           */     'o',          'O',         0,
48  /* 0x19 - 'p'           */     'p',          'P',         0,
49  /* 0x1A - '['           */     '[',          '{',         0,
50  /* 0x1B - ']'           */     ']',          '}',         0,
51  /* 0x1C - CR/LF         */     ENTER,        ENTER,       PAD_ENTER,
52  /* 0x1D - l. Ctrl       */     CTRL_L,       CTRL_L,      CTRL_R,
```



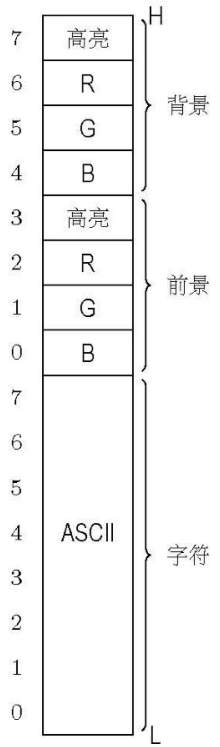
四、需要了解的一些知识

2. 显示输出

- TTY的概念:TeleTYpes
- TTY与输入输出
- 文本模式字符的显示原理




```
; GDT ------------------------------------------------------------------------------------------------------------
;                                              段基址           段界限      , 属性
LABEL_GDT:          Descriptor             0,                    0, 0                        ; 空描述符
LABEL_DESC_FLAT_C:      Descriptor             0,              0fffffh, DA_CR  | DA_32 | DA_LIMIT_4K          ; 0 ~ 4G
LABEL_DESC_FLAT_RW:     Descriptor             0,              0fffffh, DA_DRW | DA_32 | DA_LIMIT_4K          ; 0 ~ 4G
LABEL_DESC_VIDEO:       Descriptor       0B8000h,               0ffffh, DA_DRW                      | DA_DPL3    ; 显存首地址
; GDT ------------------------------------------------------------------------------------------------------------

GdtLen      equ $ - LABEL_GDT
GdtPtr      dw  GdtLen - 1              ; 段界限
        dd  BaseOfLoaderPhyAddr + LABEL_GDT     ; 基地址

; GDT 选择子 ----------------------------------------------------------------------------------------
SelectorFlatC       equ LABEL_DESC_FLAT_C   - LABEL_GDT
SelectorFlatRW      equ LABEL_DESC_FLAT_RW  - LABEL_GDT
SelectorVideo       equ LABEL_DESC_VIDEO    - LABEL_GDT + SA_RPL3
; GDT 选择子 ----------------------------------------------------------------------------------------
```



武漢大学

四、需要了解的一些知识

2. 显示输出

- 基本硬件特性

表7.4 VGA寄存器

寄存器		读端口	写端口
General Registers	Miscellaneous Output Register	0x3CC	0x3C2
	Input Status Register 0	0x3C2	-
	Input Status Register 1	0x3DA	-
	Feature Control Register	0x3CA	0x3DA
	Video Subsystem Enable Register	0x3C3	
Sequencer Registers	Address Register	0x3C4	
	Data Registers	0x3C5	
CRT Controller Registers	Address Register	0x3D4	
	Data Registers	0x3D5	
Graphics Controller Registers	Address Register	0x3CE	
	Data Registers	0x3CF	
Attribute Controller Registers	Address Register	0x3C0	
	Data Registers	0x3C1	0x3C0
Video DAC Palette Registers	Write Address	0x3C8	
	Read Address	-	0x3C7
	DAC State	0x3C7	-
	Data	0x3C9	
	Pel Mask	0x3C6	-

表7.5 CRT Controller Data Registers

寄存器名称	索引	寄存器名称	索引
Horizontal Total Register	00h	Start Address Low Register	0Dh
End Horizontal Display Register	01h	Cursor Location High Register	0Eh
Start Horizontal Blanking Register	02h	Cursor Location Low Register	0Fh
End Horizontal Blanking Register	03h	Vertical Retrace Start Register	10h
Start Horizontal Retrace Register	04h	Vertical Retrace End Register	11h
End Horizontal Retrace Register	05h	Vertical Display End Register	12h
Vertical Total Register	06h	Offset Register	13h
Overflow Register	07h	Underline Location Register	14h
Preset Row Scan Register	08h	Start Vertical Blanking Register	15h
Maximum Scan Line Register	09h	End Vertical Blanking	16h
Cursor Start Register	0Ah	CRTC Mode Control Register	17h
Cursor End Register	0Bh	Line Compare Register	18h
Start Address High Register	0Ch		

```
out_byte(0x3D4, idx);
out_byte(0x3D5, new_value);
```

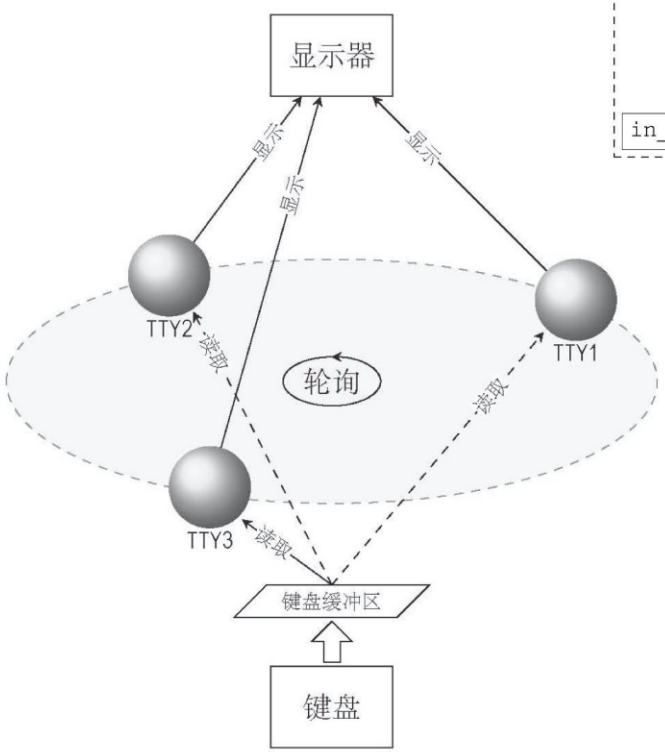
四、需要了解的一些知识

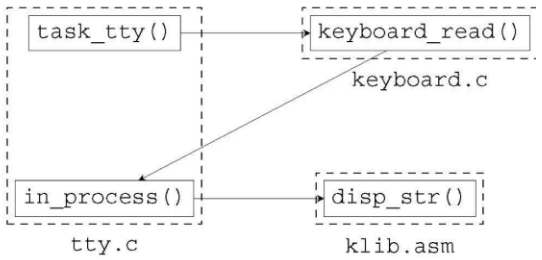
2. 显示输出
   - 基本硬件特性

```
PUBLIC void in_process(u32 key)
{
        char output[2] = {'\0', '\0'};

        if (!(key & FLAG_EXT)) {
                output[0] = key & 0xFF;
                disp_str(output);

                disable_int();
                out_byte(CRTC_ADDR_REG, CURSOR_H);
                out_byte(CRTC_DATA_REG, ((disp_pos/2)>>8)&0xFF);
                out_byte(CRTC_ADDR_REG, CURSOR_L);
                out_byte(CRTC_DATA_REG, (disp_pos/2)&0xFF);
                enable_int();
        }
        else {
                int raw_code = key & MASK_RAW;
                switch(raw_code) {
                case UP:
                        if ((key & FLAG_SHIFT_L) || (key & FLAG_SHIFT_R)) {
                                disable_int();
                                out_byte(CRTC_ADDR_REG, START_ADDR_H);
                                out_byte(CRTC_DATA_REG, ((80*15) >> 8) & 0xFF);
                                out_byte(CRTC_ADDR_REG, START_ADDR_L);
                                out_byte(CRTC_DATA_REG, (80*15) & 0xFF);
                                enable_int();
                        }
                        break;
                case DOWN:
                        if ((key & FLAG_SHIFT_L) || (key & FLAG_SHIFT_R)) {
                                /* Shift+Down, do nothing */
                        }
                        break;
                default:
                        break;
                }
        }
}
```

四、需要了解的一些知识

3. TTY多任务与多控制台





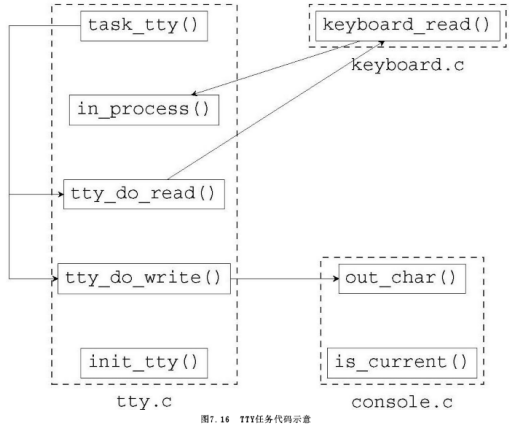
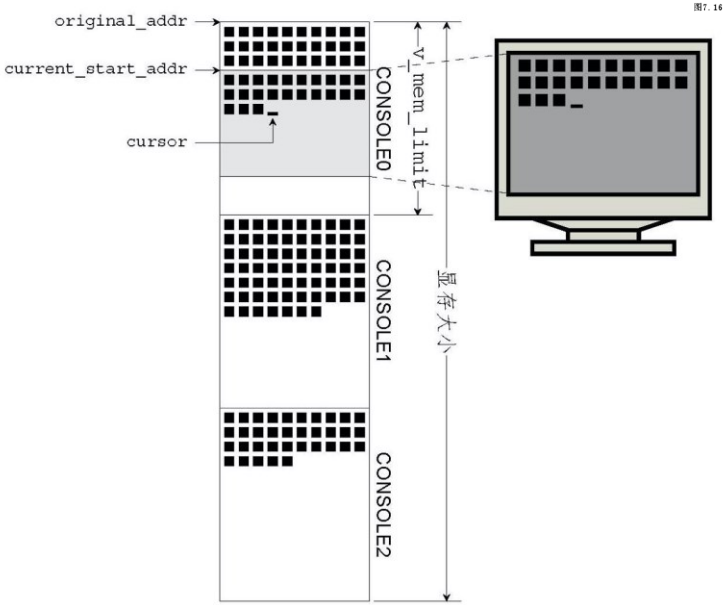


图7.16 TTY任务代码示意



四、需要了解的一些知识

4. 分层区别用户程序和系统任务

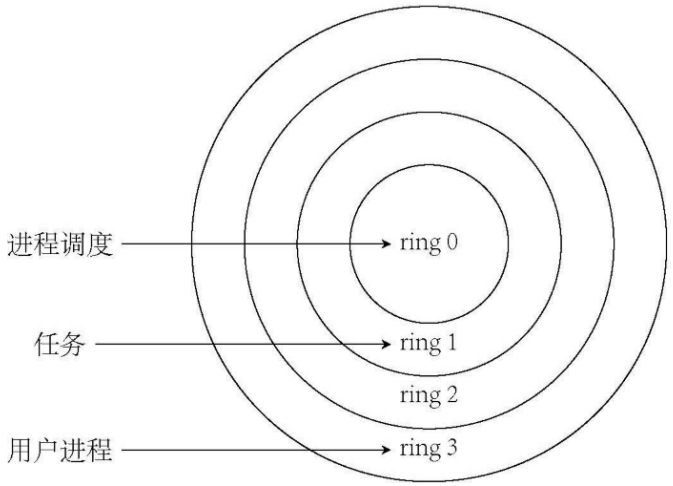


图7.24 区分任务和用户进程

代码7.48 增加NR_PROC（cha

```
51 /* Number of tasks & procs */
52 #define NR_TASKS        1
53 #define NR_PROCS        3
```

代码7.50 初始化进程表时区分task和proc（chapter7/n/kernel/main.c）

```
22 PUBLIC int kernel_main( )
23 {
    ...
31          u8              privilege;
32          u8              rpl;
33          int             eflags;
34          for (i = 0; i < NR_TASKS+NR_PROCS; i++) {
35                  if (i < NR_TASKS) {     /* 任务 */
36                          p_task = task_table + i;
37                          privilege = PRIVILEGE_TASK;
38                          rpl = RPL_TASK;
39                          eflags = 0x1202; /* IF=1, IOPL=1, bit 2 is always 1 */
40                  }
41                  else {          /* 用户进程 */
42                          p_task = user_proc_table + (i - NR_TASKS);
43                          privilege = PRIVILEGE_USER;
44                          rpl = RPL_USER;
45                          eflags = 0x202; /* IF=1, bit 2 is always 1 */
46                  }
47
48                  strcpy(p_proc->p_name, p_task->name);   //name of the process
49                  p_proc->pid = i;                        // pid
50
51                  p_proc->ldt_sel = selector_ldt;
52
53                  memcpy(&p_proc->ldts[0],  &gdt[SELECTOR_KERNEL_CS >> 3],
54                          sizeof(DESCRIPTOR));
55                  p_proc->ldts[0].attr1 = DA_C | privilege << 5;
56                  memcpy(&p_proc->ldts[1],  &gdt[SELECTOR_KERNEL_DS >> 3],
57                          sizeof(DESCRIPTOR));
58                  p_proc->ldts[1].attr1 = DA_DRW | privilege << 5;
59                  p_proc->regs.cs = (0 & SA_RPL_MASK & SA_TI_MASK) | SA_TIL | rpl;
60                  p_proc->regs.ds = (8 & SA_RPL_MASK & SA_TI_MASK) | SA_TIL | rpl;
61                  p_proc->regs.es = (8 & SA_RPL_MASK & SA_TI_MASK) | SA_TIL | rpl;
62                  p_proc->regs.fs = (8 & SA_RPL_MASK & SA_TI_MASK) | SA_TIL | rpl;
63                  p_proc->regs.ss = (8 & SA_RPL_MASK & SA_TI_MASK) | SA_TIL | rpl;
64                  p_proc->regs.gs = (SELECTOR_KERNEL_GS & SA_RPL_MASK) | rpl;
65
```

四、需要了解的一些知识

5. printf

```
52  int printf(const char *fmt, ...)
53  {
54          int i;
55          char buf[256];
56
57          va_list arg = (va_list)((char*)(&fmt) + 4); /*4是参数fmt所占堆栈中的大小*/
58          i = vsprintf(buf, fmt, arg);
59          write(buf, i);
60
61          return i;
62  }
```

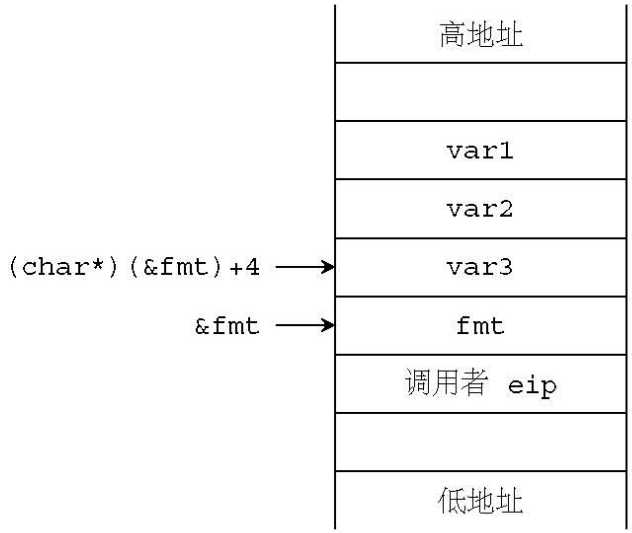


图7.25 printf调用后的堆栈情况

```
16  /*======================================================================*
17                              vsprintf
18   *======================================================================*/
19  int vsprintf(char *buf, const char *fmt, va_list args)
20  {
21      char*   p;
22      char    tmp[256];
23      va_list p_next_arg = args;
24
25      for (p=buf;*fmt;fmt++) {
26          if (*fmt != '%') {
27              *p++ = *fmt;
28              continue;
29          }
30
31          fmt++;
32
33          switch (*fmt) {
34          case 'x':
35              itoa(tmp, *((int*)p_next_arg));
36              strcpy(p, tmp);
37              p_next_arg += 4;
38              p += strlen(tmp);
39              break;
40          case 's':
41              break;
42          default:
43              break;
44          }
45      }
46
47      return (p - buf);
48  }
```

四、需要了解的一些知识

5. printf

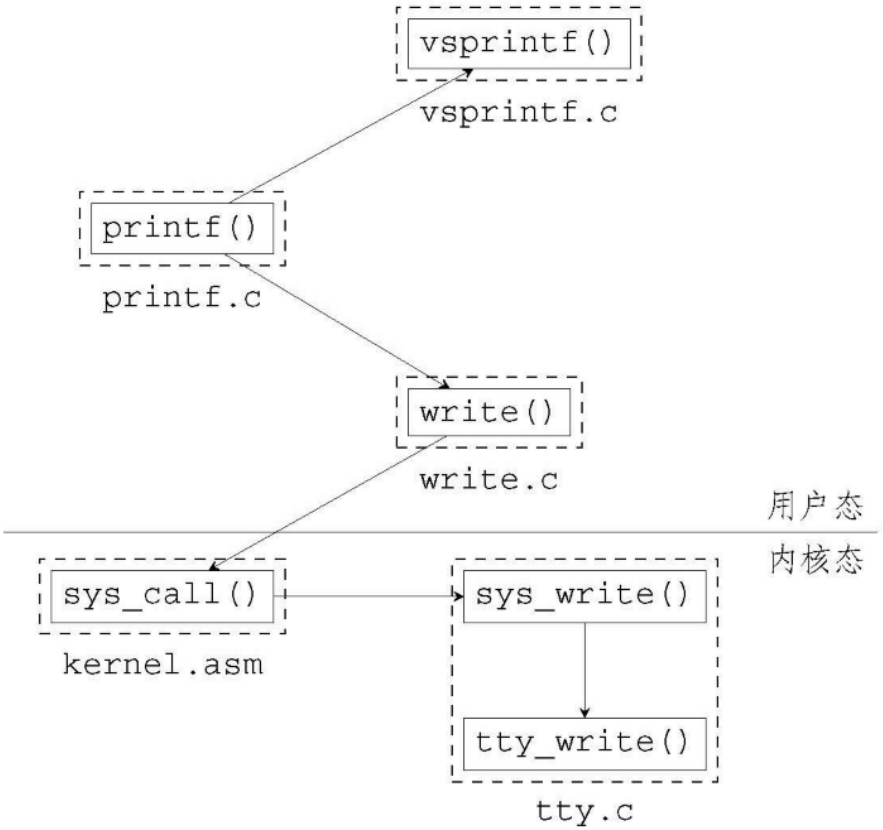


图7.28 printf调用过程示意图

谢　谢！