

《计算机与操作系统》第四次实验

1. 实验代码

1.1 读写公平

```
1  void read_fair(int slices) {
2      P(&S);
3
4      P(&reader_count_mutex);
5      P(&reader_mutex);
6      if (++readers == 1)
7          P(&rw_mutex); // 有读者，禁止写
8      V(&reader_mutex);
9
10     V(&S);
11
12     read_proc(slices);
13
14     P(&reader_mutex);
15     if (--readers == 0)
16         V(&rw_mutex); // 没有读者，可以开始写了
17     V(&reader_mutex);
18     V(&reader_count_mutex);
19 }
20
21 void write_fair(int slices) {
22     P(&S);
23     P(&rw_mutex);
24     write_proc(slices);
25     V(&rw_mutex);
26     V(&S);
27 }
```

1.2 读者优先

```
1
2  void read_rf(int slices) {
3      P(&reader_count_mutex);
4      P(&reader_mutex);
5      if (++readers == 1)
6          P(&writer_mutex); // 有读者时不允许写
7      V(&reader_mutex);
8
9      read_proc(slices);
10
11     P(&reader_mutex);
12     if (--readers == 0)
13         V(&writer_mutex); // 没有读者时可以开始写
14     V(&reader_mutex);
15     V(&reader_count_mutex);
16 }
17
18 void write_rf(int slices) {
19     P(&writer_mutex);
```

```

20     write_proc(slices);
21     V(&writer_mutex);
22 }

```

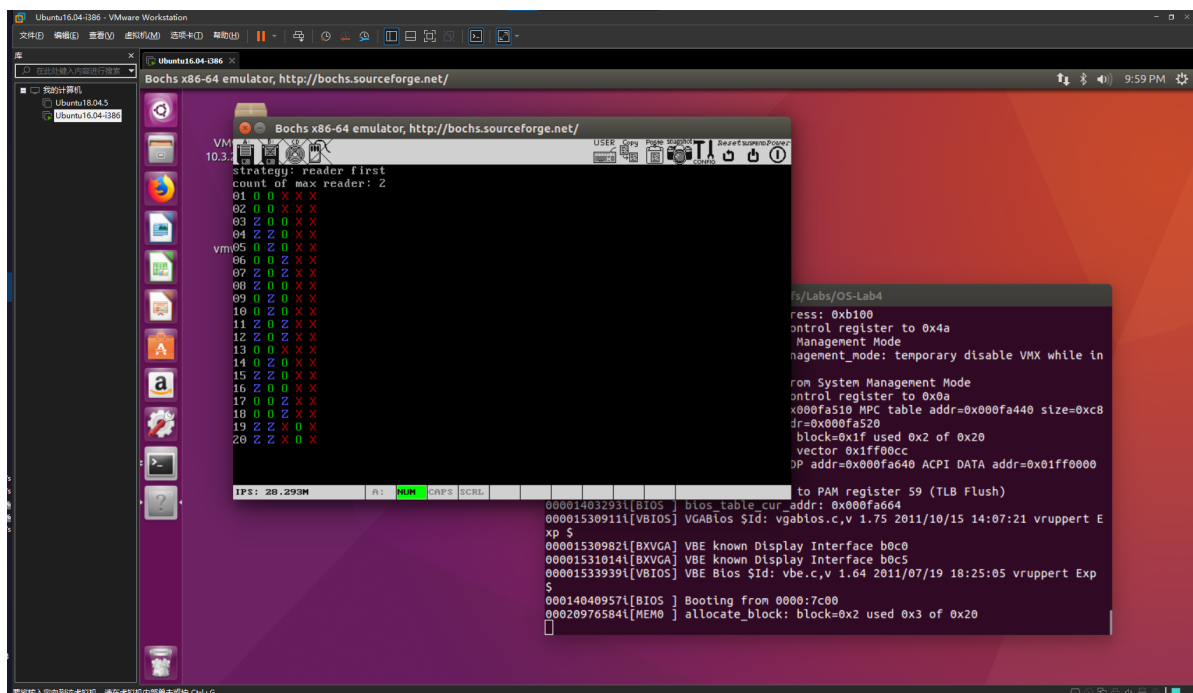
1.3 写者优先

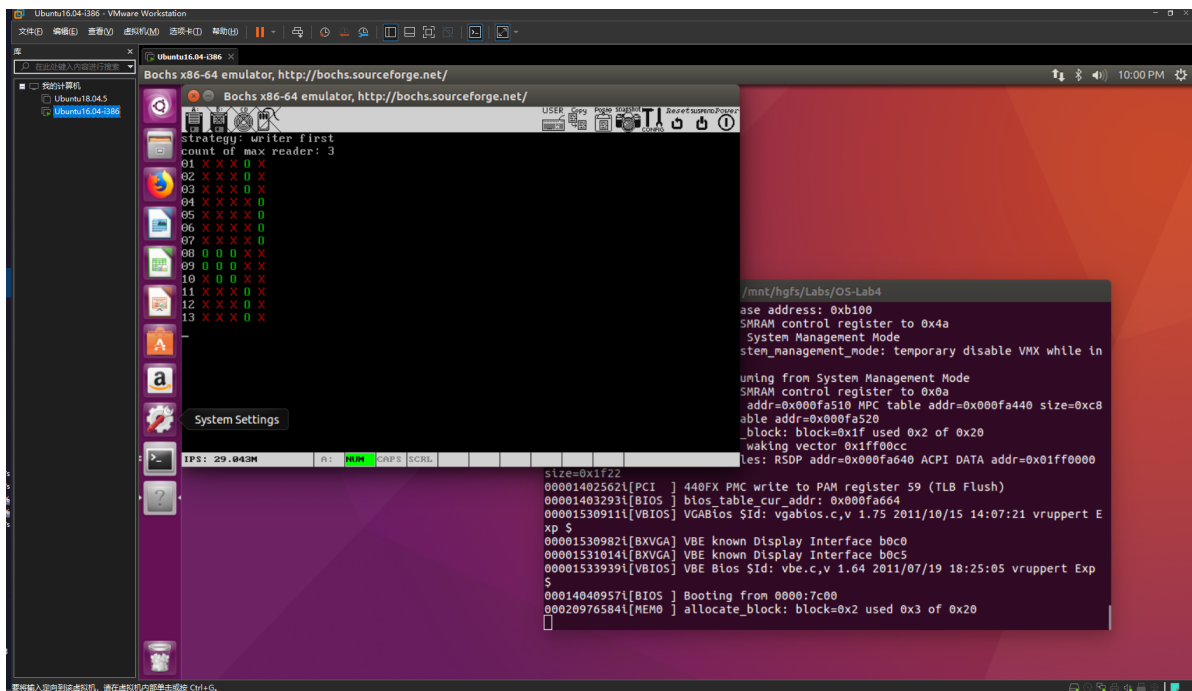
```

1
2 void read_wf(int slices) {
3     P(&rw_mutex);
4     P(&reader_mutex);
5     if (++readers == 1)
6         P(&writer_mutex); // 有读者时不允许写
7     V(&reader_mutex);
8     V(&rw_mutex);
9
10    P(&reader_count_mutex);
11    read_proc(slices);
12    V(&reader_count_mutex);
13
14    P(&reader_mutex);
15    if (--readers == 0)
16        V(&writer_mutex); // 没有读者时可以开始写
17    V(&reader_mutex);
18 }
19
20 void write_wf(int slices) {
21     P(&rw_mutex);
22     P(&writer_mutex);
23     write_proc(slices);
24     V(&writer_mutex);
25     V(&rw_mutex);
26 }

```

2. 实验截图





3. 实验问题

3.1 进程是什么？

进程是计算机中的程序关于某个数据集合上的一次运行活动，是系统**进行资源分配和调度的基本单位**。从宏观来看，它有自己的目标（功能），同时又能受控于进程调度模块；从微观来看，它可以利用系统的资源，有自己的代码和数据，同时拥有自己的堆栈，进程需要被调度。

3.2 进程表是什么？

进程表是存储进程状态信息的数据结构。进程表是进程存在的唯一标识，是操作系统用来记录和刻画进程状态及环境信息的数据结构，是进程动态特征的汇集，也是操作系统掌握进程的唯一资料结构和管理进程的主要依据。

3.3 进程栈是什么？

进程运行时自身的堆栈。

3.4 当寄存器的值已经被保存到进程表内，esp 应该指向何处来避免破坏进程表的值？

进程运行时，esp 指向堆栈中的某个位置。寄存器的值刚刚被保存到进程表内，esp 是指向进程表某个位置的。如果接下来进行任何的堆栈操作，都会破坏掉进程表的值。为解决这个问题，使用内核栈，让 esp 指向内核栈。

3.5 tty 是什么？

Teletype 的缩写。终端是一种字符型设备，它有多种类型，通常使用 TTY 来简称各种类型的终端设备。不同 TTY 对应的输入设备是同一个键盘。

3.6 不同的 tty 为什么输出不同的画面在同一个显示器上？

不同 TTY 各有一个 CONSOLE，各个 CONSOLE 公用同一块显存。虽然不同的 TTY 对应的输入设备是同一个键盘，但输出却好比是在不同的显示器上，因为不同的 TTY 对应的屏幕画面可能是迥然不同的。实际上，我们当然是在使用同一个显示器，画面的不同只不过是显示了显存的不同位置罢了。

3.7 解释 tty 任务执行的过程？

在 TTY 任务中执行一个循环，这个循环将轮询每一个 TTY，处理它的事件，包括从键盘缓冲区读取数据、显示字符等内容。轮询到每一个 TTY 时：

1. 处理输入：查看其是否为当前 TTY。只有当某个 TTY 对应的控制台是当前控制台时，它才可以读取键盘缓冲区；
2. 处理输出：如果有要显示的内容则显示它。

3.8 tty 结构体中大概包括哪些内容？

```

1  #define TTY_IN_BYTES    256    /* tty input S size */
2
3  struct s_console;
4
5  /* TTY */
6  typedef struct {
7      u32 in_buf[TTY_IN_BYTES]; /* TTY 输入缓冲区 */
8      u32 *p_inbuf_head;        /* 指向缓冲区中下一个空闲位置 */
9      u32 *p_inbuf_tail;        /* 指向键盘任务应处理的键值 */
10     int inbuf_count;           /* 缓冲区中已经填充了多少 */
11
12     struct s_console *p_console;
13 } TTY;
```

3.9 console 结构体中大概包括哪些内容？

```

1  /* CONSOLE */
2  typedef struct s_console {
3      unsigned int current_start_addr; /* 当前显示到了什么位置 */
4      unsigned int original_addr;      /* 当前控制台对应显存位置 */
5      unsigned int v_mem_limit;        /* 当前控制台占的显存大小 */
6      unsigned int cursor;             /* 当前光标位置 */
7      u8 color;
8  } CONSOLE;
9
10 #define SCR_UP    1    /* scroll forward */
11 #define SCR_DN    -1   /* scroll backward */
12
13 #define SCREEN_SIZE    (80 * 25)
14 #define SCREEN_WIDTH    80
15
16 #define DEFAULT_CHAR_COLOR    0x07    /* 0000 0111 黑底白字 */
```

3.10 什么是时间片？

3.11 结合实验代码解释什么是内核函数？什么是系统调用？