

实现系统调用

```
1 PUBLIC  system_call sys_call_table[NR_SYS_CALL] = {
2         sys_get_ticks,
3         sys_write_str,
4         sys_sleep,
5         p_process,
6         v_process
7 };
```

其中，`sleep`是封装好的`milli_second`，`write_str`是封装好到用户态的`disp_str`。

读者写者问题实现

对于读写处理的函数集合：

```
1 read_f read_funcs[3] = {read_v0, read_rf, read_wf};
2 write_f write_funcs[3] = {write_v0, write_rf, write_wf};
```

读者：

```
1  /*=====
2  =====*
3  *=====
4  =====*/
5  void ReaderA()
6  {
7      //休息一个时间片开始进行
8      sleep_ms(TIME_SLICE);
9      while(1){
10         //读者A, 消耗2个时间片, 用颜色"\01"表示
11         read_funcs[strategy]('A', 2, '\01');
12         sleep_ms(TIME_SLICE);
13     }
```

```

13 }
14
15 /*=====
=====*
16
17 ReaderB
18 *=====
=====*/
19 void ReaderB()
20 {
21     //休息两个个时间片开始进行
22     sleep_ms(2*TIME_SLICE);
23     while(1){
24         //读者B，消耗3个时间片，用颜色"\02"表示
25         read_funcs[strategy]('B', 3, '\02');
26         sleep_ms(TIME_SLICE);
27     }
28 }
29
30 /*=====
=====*
31
32 ReaderC
33 *=====
=====*/
34 void ReaderC()
35 {
36     //休息三个时间片开始进行
37     sleep_ms(3*TIME_SLICE);
38     while(1){
39         //读者C，消耗3个时间片，用颜色"\03"表示
40         read_funcs[strategy]('C', 3, '\03');
41         sleep_ms(TIME_SLICE);
42     }
43 }

```

写者:

```

1 /*=====
=====*
2
3 WriterD

```

```

3  *=====
   =====*/
4  void writerD()
5  {
6      //休息四个时间片开始进行
7      sleep_ms(4*TIME_SLICE);
8      while(1){
9          //写者D, 消耗3个时间片, 用颜色"\04"表示
10         write_funcs[strategy]('D', 3, '\04');
11         sleep_ms(TIME_SLICE);
12     }
13 }
14
15 /*=====
   =====*
16                                     WriterE
17  *=====
   =====*/
18 void writerE()
19 {
20     //休息五个时间片开始进行
21     sleep_ms(5*TIME_SLICE);
22     while(1){
23         //写者E, 消耗3个时间片, 用颜色"\04"表示
24         write_funcs[strategy]('E', 4, '\05');
25         sleep_ms(TIME_SLICE);
26     }
27 }

```

读者优先

```

1  void read_rf(char proc, int slices, char color){
2      printf("%c%c arrives\n", color, proc);
3
4      P(&r_mutex);
5      if (readers==0)
6          P(&rw_mutex);
7      readers++;

```

```

8      V(&r_mutex);
9
10     read_proc(proc, slices, color);
11
12     P(&r_mutex);
13     tr--;
14     V(&r_mutex);
15
16     V(&n_r_mutex);
17
18     P(&r_mutex);
19     readers--;
20     if (readers==0)
21         V(&rw_mutex); // 没有读者，可以开始写了
22     V(&r_mutex);
23
24 }
25
26 void write_rf(char proc, int slices, char color){
27     printf("%c%c arrives\n", color, proc);
28     P(&rw_mutex);
29     writing = 1;
30     // 写过程
31     write_proc(proc, slices, color);
32     writing = 0;
33     V(&rw_mutex);
34 }

```

运行截图

ABC三个读者的优先级高于写者，在第一个时间段内A进入，后续BC进入使得读者一直在进行，不会被写者抢，故在图中可以看到写者D和E始终无法进行，一直是在读。

Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

A: B: CD

USER Copy Paste Snapshot

```
A is reading
time: 0
1 of processes are reading
B arrives
B starts reading
B is reading
time: 1
2 of processes are reading
A finishes reading
C arrives
C starts reading
C is reading
time: 2
2 of processes are reading
A arrives
A starts reading
A is reading
D arrives
time: 3
3 of processes are reading
B finishes reading
E arrives
time: 4
2 of processes are reading
```

Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

A: B: CD

USER Copy Paste Snapshot

```
C starts reading
C is reading
time: 6
3 of processes are reading
time: 7
3 of processes are reading
A finishes reading
B finishes reading
time: 8
1 of processes are reading
A arrives
A starts reading
A is reading
B arrives
B starts reading
B is reading
C finishes reading
time: 9
2 of processes are reading
C arrives
C starts reading
C is reading
time: A
3 of processes are reading
```

IPS: 56.667M A: NUM CAPS SCRL

写者优先

```
1 void read_wf(char proc, int slices, char color){
2     printf("%c%c arrives\n", color, proc);
3     P(&n_r_mutex);
4     P(&queue);
5     P(&r_mutex);
6     if (readers==0)
7         P(&rw_mutex);
8     readers++;
9     V(&r_mutex);
10    V(&queue);
11    //读过程开始
12    read_proc(proc, slices, color);
13    P(&r_mutex);
14    readers--;
15    if (readers==0)
16        V(&rw_mutex); // 没有读者, 可以开始写了
17    V(&r_mutex);
18    V(&n_r_mutex);
19 }
20
21 void write_wf(char proc, int slices, char color){
22     printf("%c%c arrives\n", color, proc);
23     P(&w_mutex);
24     // 写过程
25     if (writers==0)
26         P(&queue);
27     writers++;
28     V(&w_mutex);
29
30     P(&rw_mutex);
31     writing = 1;
32     write_proc(proc, slices, color);
33     writing = 0;
34     V(&rw_mutex);
35
36     P(&w_mutex);
```

```

37     writers--;
38     if (writers==0)
39         V(&queue);
40     V(&w_mutex);
41 }

```

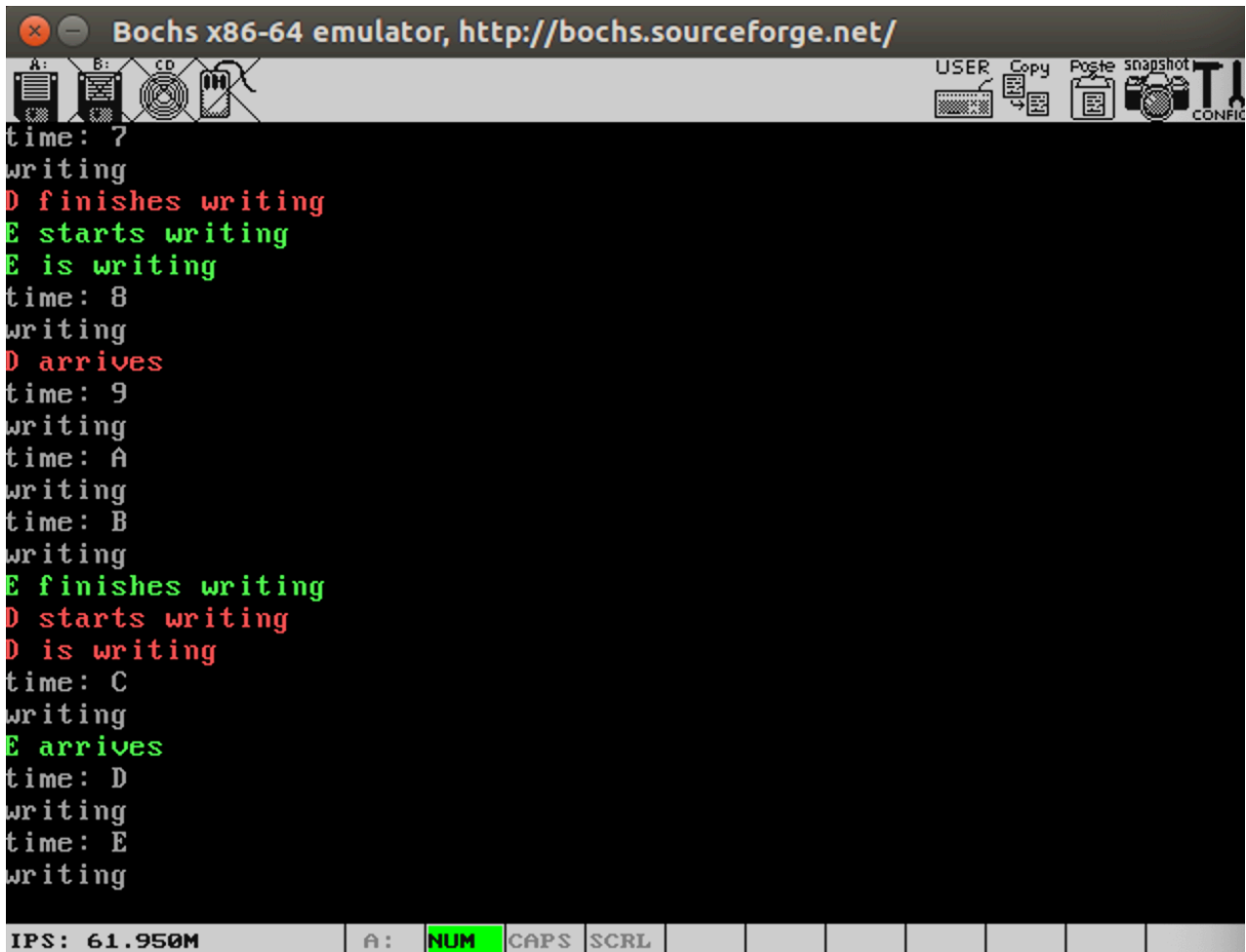
截图

前三个读进程结束之后，DE写进程到来，因为写者优先，在第五个时刻AC都结束读，B和DE在等待，DE开始写，如下5到7都是D在进行写。8~10是E进行写，后续都是DE轮流进行读写。

```

Bochs x86-64 emulator, http://bochs.sourceforge.net/
2 of processes are reading
A arrives
A starts reading
A is reading
D arrives
time: 3
3 of processes are reading
B finishes reading
E arrives
time: 4
2 of processes are reading
A finishes reading
B arrives
C finishes reading
D starts writing
D is writing
time: 5
writing
A arrives
C arrives
time: 6
writing
time: 7
writing
IPS: 65.360M  A: NUM  CAPS  SCRL

```



Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

time: 7
writing
D finishes writing
E starts writing
E is writing
time: 8
writing
D arrives
time: 9
writing
time: A
writing
time: B
writing
E finishes writing
D starts writing
D is writing
time: C
writing
E arrives
time: D
writing
time: E
writing

IPS: 61.950M A: NUM CAPS SCRL

读写公平：防止饿死

```
1 void read_gp(char proc, int slices, char color){
2     printf("%c%c arrives\n", color, proc);
3     P(&queue);
4     P(&n_r_mutex);
5     P(&r_mutex);
6     if (readers==0)
7         P(&rw_mutex); // 有读者，禁止写
8     readers++;
9     V(&r_mutex);
10    V(&queue);
11    read_proc(proc, slices, color);
12    P(&r_mutex);
13    readers--;
14    if (readers==0)
15        V(&rw_mutex); // 没有读者，可以开始写了
16    V(&r_mutex);
17    V(&n_r_mutex);
```


Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

A: B: CD

USER Copy Paste snapshot

```
A is reading
time: 0
1 of processes are reading
B arrives
B starts reading
B is reading
time: 1
2 of processes are reading
A finishes reading
C arrives
C starts reading
C is reading
time: 2
2 of processes are reading
A arrives
A starts reading
A is reading
D arrives
time: 3
3 of processes are reading
B finishes reading
E arrives
time: 4
2 of processes are reading
```

IPS: 63.032M A: NUM CAPS SCRL