

本次实验使用 Orange's 第 8 章源码，在此基础上修改完成，并参考了往年学长学姐的代码。

1、添加系统调用

(1) 在 proto.h 里添加函数声明，这里是用户调用的函数声明

```
104 /* 系统调用 - 用户级 */
105 PUBLIC int sendrec(int function, int src_dest, MESSAGE* p_msg);
106 PUBLIC int printx(char* str);
107 PUBLIC int sleep(int milli_sec);
108 PUBLIC int p(struct semaphore * s);
109 PUBLIC int v(struct semaphore * s);
110
126 /* system call */
127 #define NR_SYS_CALL 5
```

将 NR_SYS_CALL 设置为 5

(2) 在 syscall.asm 里设置调用号和调用的 nasm 代码

```
INT_VECTOR_SYS_CALL equ 0x90
NR_printx equ 0
NR_sendrec equ 1
NR_sleep equ 2
NR_P equ 3
NR_V equ 4
```

nasm 代码模仿书上的格式：

```
43 ret
44 sleep:
45     mov     eax, NR_sleep
46     mov     edx, [esp + 4]
47     int     INT_VECTOR_SYS_CALL
48     ret
49
```

(3) 在 global.c 系统调用表中加入新的系统调用

```
42 PUBLIC system_call sys_call_table[NR_SYS_CALL] = {sys_printx,
43     sys_sendrec, sys_process_sleep, sys_tem_p, sys_tem_v};
```

(4) 在 proto.h 里添加系统调用的函数申明，在 kernel.asm 里 syscall 传入了四个参数，这里不需要全部用到，最后一个参数是指向当前进程的指针。

```
94 /* 系统调用 - 系统级 */
95 /* proc.c */
96 PUBLIC int sys_sendrec(int function, int src_dest, MESSAGE* m, struct proc* p);
97 PUBLIC int sys_printx(int unused1, int unused2, char* s, struct proc * p_proc);
98 PUBLIC void sys_process_sleep(int unused1, int unused2, int milli_sec, struct proc * p);
99 PUBLIC void sys_tem_p(int unused1, int unused2, struct semaphore * s, struct proc * p);
100 PUBLIC void sys_tem_v(int unused1, int unused2, struct semaphore * s, struct proc * p);
```

其中 semaphore 是信号量的数据结构，定义在 proc.h 文件里，这里使用数组代表等待队列：

```
75 struct semaphore
76 {
77     int value;
78     int len;
79     struct proc * list[10];
80 };
```

(5) 系统调用函数的实现

```
37 PUBLIC void sys_process_sleep(int unused1, int unused2, int milli_sec, struct proc * p) {
38     int tick_time = milli_sec * HZ / 1000;
39     setSleep_ticks(tick_time, p);
40 }
```

sleep: p->sleep_ticks = ticks; 设置进程的 sleep_ticks，在进程调度时若 sleep 不为 0 则不参与进程调度，但会将 sleep_ticks 减 1，直到为 0 时重新参与进程调度。

p、v 操作实现在 main.c 里，因为中断的时候这里不允许再中断，所以 pv 操作就变成了原子操作

p 操作：信号量减 1，若小于 0 则阻塞自己移入等待队列，开始下一次进程调度

```

22     s->value--;
23     if (s->value<0) {
24         s->list[s->len++] = p_proc_ready;
25         p_proc_ready->p_flags = 1;
26         schedule();
27     }

```

v 操作：信号量加 1，若小于等于 0 则从等待队列释放一个进程

```

s->value++;
if (s->value<=0) {
    s->list[0]->p_flags = 0;
    int i = 1;
    for (i=1;i<=s->len;++i) {
        s->list[i-1] = s->list[i];
    }
    s->len--;
}

```

print 系统调用的代码原来就有，在 tty.c 里：

```

184  /*=====
185  |               |               |               |               |               |               |
186  *=====
187  PUBLIC int sys_printx(int _unused1, int _unused2, char* s, struct
188  {
189      const char * p;

```

封装好的 printf 在 printf.c 里：

```

60  int printf(const char *fmt, ...) {
61      int i;
62      char buf[256];
63
64      va_list arg = (va_list)((char*)&fmt + 4); /*4是参数fmt所占堆栈中的大
65      i = vsprintf(buf, fmt, arg);
66      buf[i] = 0;
67      printx(buf);
68
69      return i;
70  }

```

2、睡眠理发师问题

(1) 添加顾客进程：

在 global.c 的用户进程添加 D、E

```

27  PUBLIC struct task user_proc_table[NR_PROCS] = {
28      {TestB, STACK_SIZE_TESTB, "Barber"},
29      {TestC, STACK_SIZE_TESTC, "Customer1"},
30      {TestD, STACK_SIZE_TESTD, "Customer2"},
31      {TestE, STACK_SIZE_TESTE, "Customer3"}
32  };

```

proc.h 里的修改，修改任务数量

```

85  #define NR_TASKS      3
86  #define NR_PROCS      4

```

定义任务堆栈和堆栈大小

```

91 #define STACK_SIZE_TTY      0x8000
92 #define STACK_SIZE_SYS      0x8000
93 #define STACK_SIZE_TESTA    0x8000
94 #define STACK_SIZE_TESTB    0x8000
95 #define STACK_SIZE_TESTC    0x8000
96 #define STACK_SIZE_TESTD    0x8000
97 #define STACK_SIZE_TESTE    0x8000
98
99 #define STACK_SIZE_TOTAL    (STACK_SIZE_TTY + \
100     STACK_SIZE_SYS + \
101     STACK_SIZE_TESTA + \
102     STACK_SIZE_TESTB + \
103     STACK_SIZE_TESTC+STACK_SIZE_TESTD+STACK_SIZE_TESTE\
104     )

```

添加函数申明 (proto.h) :

```

39 PUBLIC void TestC();           //customers1 proc
40 PUBLIC void TestD();           //customers2 proc
41 PUBLIC void TestE();           //customer33 proc
42 PUBLIC void TestB(char *first);

```

(2) 具体实现理发师问题

global.c 里定义全局变量：等待人数，椅子数和信号量

```

44 PUBLIC int waiting = 0;
45 PUBLIC int CHAIR = 3;
46 PUBLIC struct semaphore customers,barbers,mutex;

```

main.c 的 kernal_main () 里初始化变量

```

120     mutex.value = 1;
121     waiting = 0;
122     CHAIR = 2;

```

理发师进程：

```

164 void TestB() {
165     printf("\n");
166     while (1) {
167         p(&customers);
168         p(&mutex);
169         waiting--;
170         printf("barber cutting!\n");
171         v(&mutex);
172         v(&barbers);
173         goToSleep(60);
174     }
175 }

```

解释：首先查看是否有顾客，没有则阻塞自己，有则进入临界区，将等待人数减 1，退出临界区，开始理发
顾客进程：


```

204▼ void TestD() {
205     int i = 1;
206     while (1) {
207▼     /* assert(0); */
208         p(&mutex);
209         printf("customer  %d  come!\n",++customerNum);
210         i = customerNum;
211▼     if (waiting < CHAIR) {
212         printf("customer  %d  wait!\n",i,waiting);
213         waiting++;
214         v(&customers);
215         v(&mutex);
216         p(&barbers);
217         printf("customer  %d  get service!\n",i);
218▼     } else {
219         v(&mutex);
220         printf("not enough chairs!-----\n");
221     }
222     printf("customer  %d  leaves\n",i);
223     ++i;
224     goToSleep(5);
225 }
226 }

```

解释：进入临界区，如果有椅子则等待，增加顾客数，申请理发师，否则直接离开。

3、输出颜色改变

在 console.c 的 outchar 函数里判断进程号

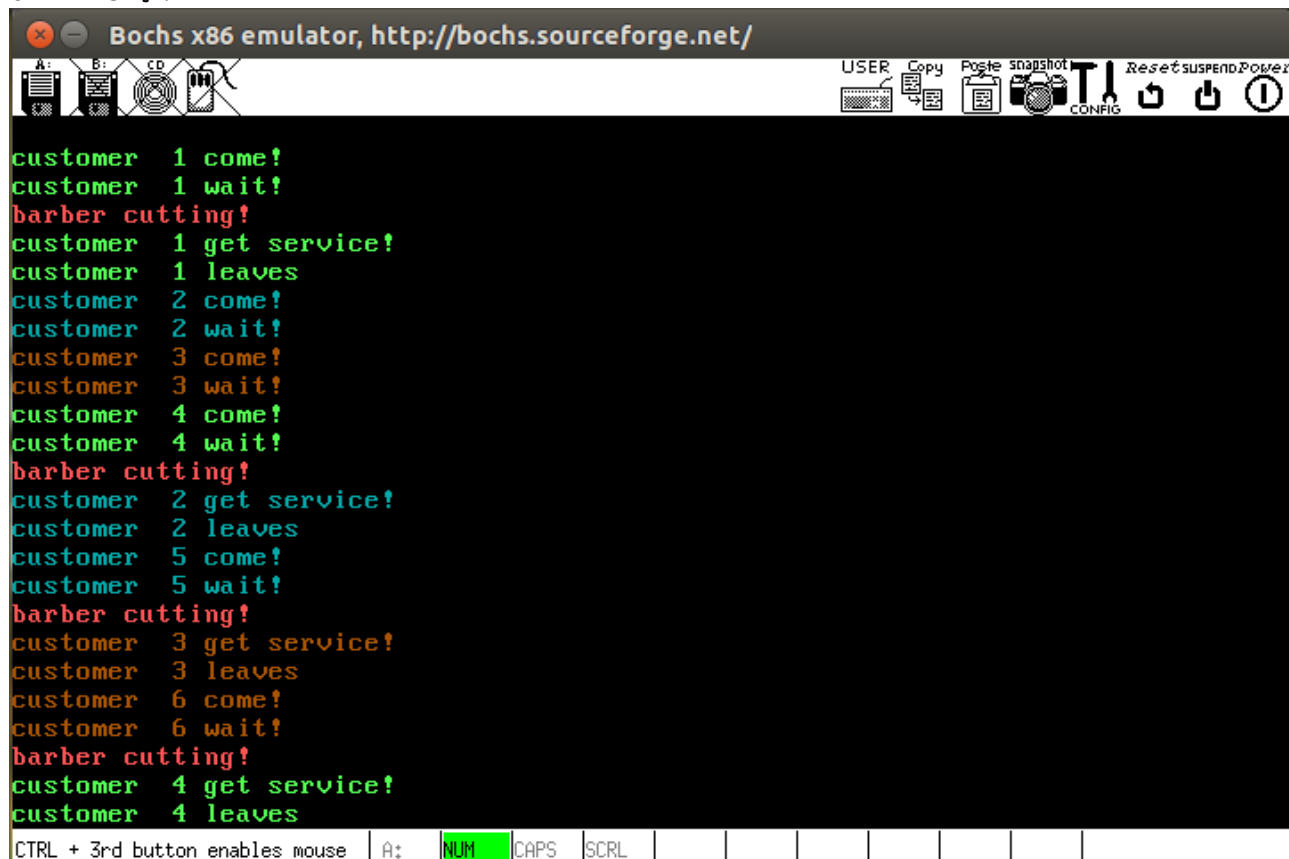
```

93 PUBLIC void out_char(CONSOLE* p_con, char ch)
94 {
95     char ch_color = DEFAULT_CHAR_COLOR;
96     switch(p_proc_ready->pid) {
97     case 3:
98         ch_color = 0x0C;
99         break;
100    case 4:
101        ch_color = 0X0A;
102        break;
103    case 5:
104        ch_color = 0X03;
105        break;
106    case 6:
107        ch_color = 0X06;
108        break;
109    }

```

输出结果：

CHAIR=3 时：



```
customer 1 come!
customer 1 wait!
barber cutting!
customer 1 get service!
customer 1 leaves
customer 2 come!
customer 2 wait!
customer 3 come!
customer 3 wait!
customer 4 come!
customer 4 wait!
barber cutting!
customer 2 get service!
customer 2 leaves
customer 5 come!
customer 5 wait!
barber cutting!
customer 3 get service!
customer 3 leaves
customer 6 come!
customer 6 wait!
barber cutting!
customer 4 get service!
customer 4 leaves
```

CHAIR=2 时：



```
not enough chairs!-----
customer 20 leaves
barber cutting!
customer 17 get service!
customer 17 leaves
customer 21 come!
customer 21 wait!
customer 22 come!
not enough chairs!-----
customer 22 leaves
barber cutting!
customer 23 come!
customer 23 wait!
customer 19 get service!
customer 19 leaves
customer 24 come!
not enough chairs!-----
customer 24 leaves
barber cutting!
customer 25 come!
customer 25 wait!
customer 21 get service!
customer 21 leaves
customer 26 come!
not enough chairs!-----
```

CHAIR=1 时：

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

A: B: CD

USER Copy Paste Snapshot T CONFIG Reset SUSPEND Power

```
customer 1 come!
customer 1 wait!
barber cutting!
customer 1 get service!
customer 1 leaves
customer 2 come!
customer 2 wait!
customer 3 come!
not enough chairs!-----
customer 3 leaves
customer 4 come!
not enough chairs!-----
customer 4 leaves
barber cutting!
customer 2 get service!
customer 2 leaves
customer 5 come!
customer 5 wait!
customer 6 come!
not enough chairs!-----
customer 6 leaves
customer 7 come!
not enough chairs!-----
customer 7 leaves
```

CTRL + 3rd button enables mouse | A: | NUM | CAPS | SCRL | | | | | | | |