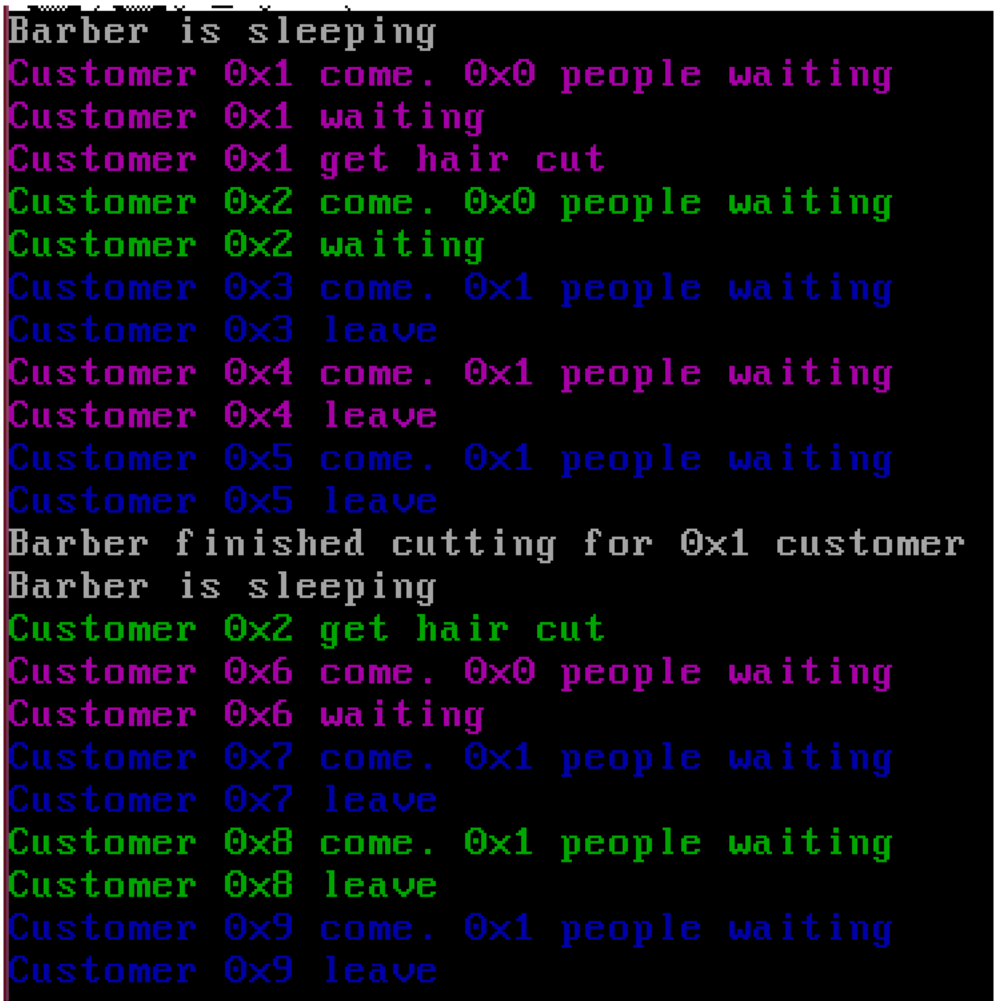
实验四说明文档

运行环境:Linux(Ubuntu 16.10 i386) 32 位

功能点覆盖及运行截图：

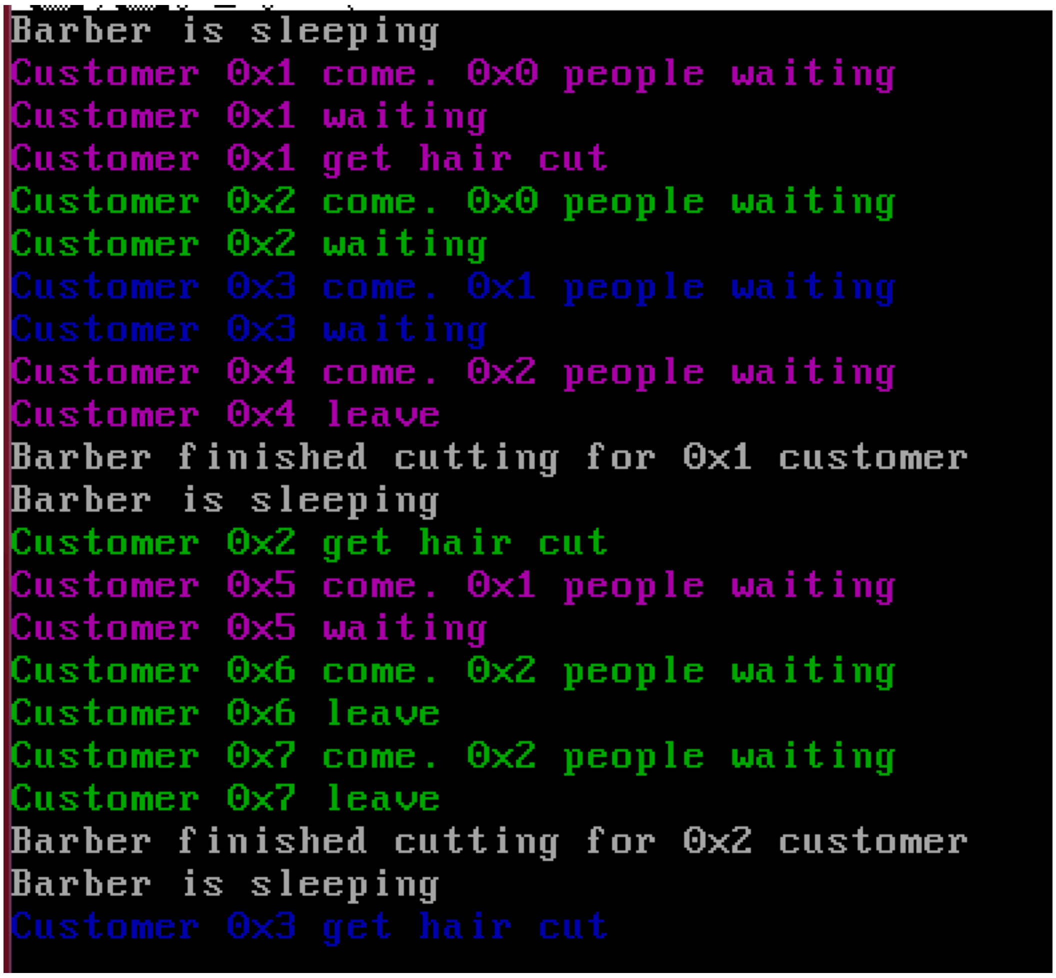
1、完成了所有要求，包括椅子数目分别为 1，2，3 的情况

2、运行截图如下：



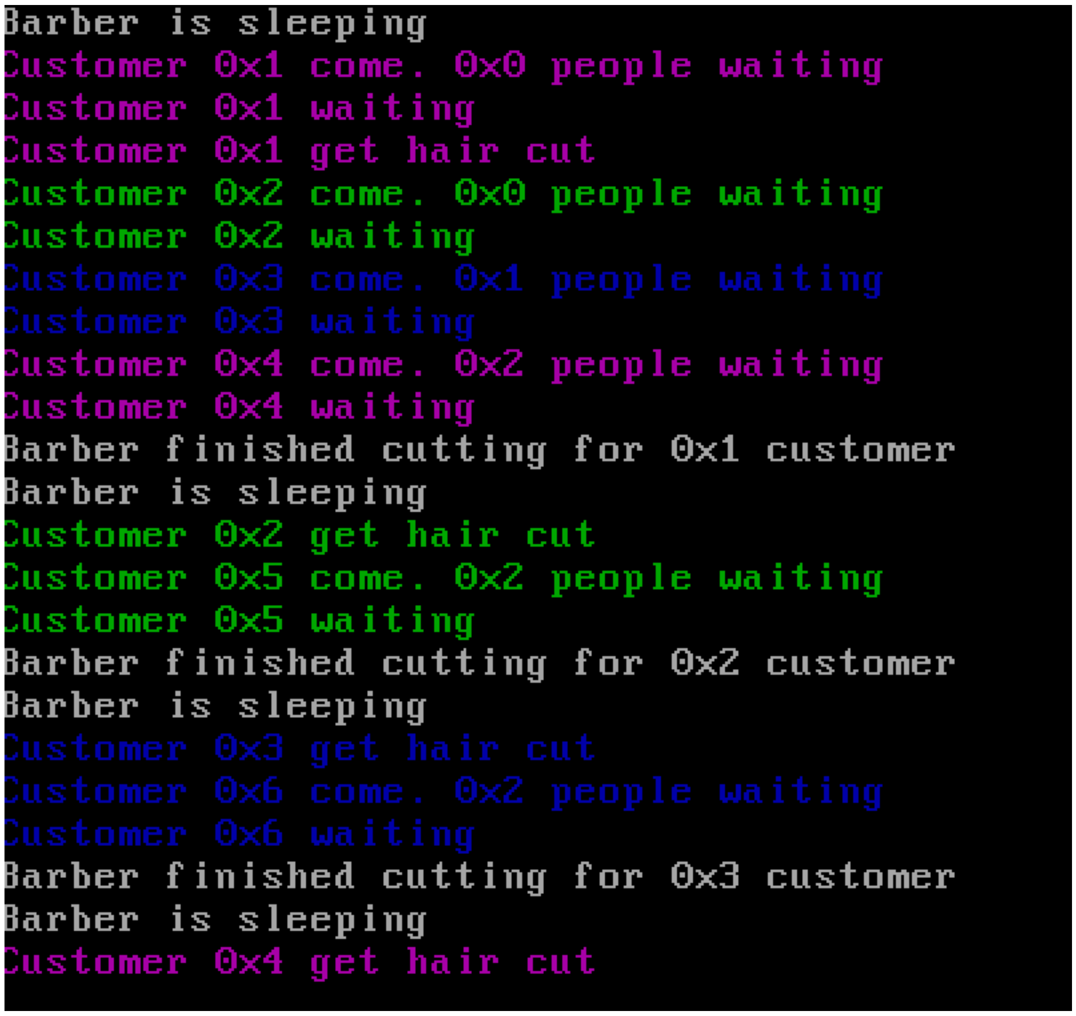
```
Barber is sleeping
Customer 0x1 come. 0x0 people waiting
Customer 0x1 waiting
Customer 0x1 get hair cut
Customer 0x2 come. 0x0 people waiting
Customer 0x2 waiting
Customer 0x3 come. 0x1 people waiting
Customer 0x3 leave
Customer 0x4 come. 0x1 people waiting
Customer 0x4 leave
Customer 0x5 come. 0x1 people waiting
Customer 0x5 leave
Barber finished cutting for 0x1 customer
Barber is sleeping
Customer 0x2 get hair cut
Customer 0x6 come. 0x0 people waiting
Customer 0x6 waiting
Customer 0x7 come. 0x1 people waiting
Customer 0x7 leave
Customer 0x8 come. 0x1 people waiting
Customer 0x8 leave
Customer 0x9 come. 0x1 people waiting
Customer 0x9 leave
```

（1 把椅子）



```
Barber is sleeping
Customer 0x1 come. 0x0 people waiting
Customer 0x1 waiting
Customer 0x1 get hair cut
Customer 0x2 come. 0x0 people waiting
Customer 0x2 waiting
Customer 0x3 come. 0x1 people waiting
Customer 0x3 waiting
Customer 0x4 come. 0x2 people waiting
Customer 0x4 leave
Barber finished cutting for 0x1 customer
Barber is sleeping
Customer 0x2 get hair cut
Customer 0x5 come. 0x1 people waiting
Customer 0x5 waiting
Customer 0x6 come. 0x2 people waiting
Customer 0x6 leave
Customer 0x7 come. 0x2 people waiting
Customer 0x7 leave
Barber finished cutting for 0x2 customer
Barber is sleeping
Customer 0x3 get hair cut
```

（2 把椅子）



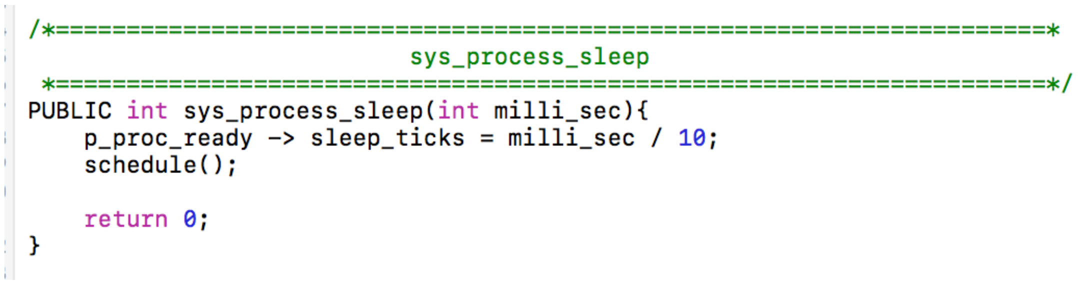
```
Barber is sleeping
Customer 0x1 come. 0x0 people waiting
Customer 0x1 waiting
Customer 0x1 get hair cut
Customer 0x2 come. 0x0 people waiting
Customer 0x2 waiting
Customer 0x3 come. 0x1 people waiting
Customer 0x3 waiting
Customer 0x4 come. 0x2 people waiting
Customer 0x4 waiting
Barber finished cutting for 0x1 customer
Barber is sleeping
Customer 0x2 get hair cut
Customer 0x5 come. 0x2 people waiting
Customer 0x5 waiting
Barber finished cutting for 0x2 customer
Barber is sleeping
Customer 0x3 get hair cut
Customer 0x6 come. 0x2 people waiting
Customer 0x6 waiting
Barber finished cutting for 0x3 customer
Barber is sleeping
Customer 0x4 get hair cut
```

（3 把椅子）

3、改动之处：

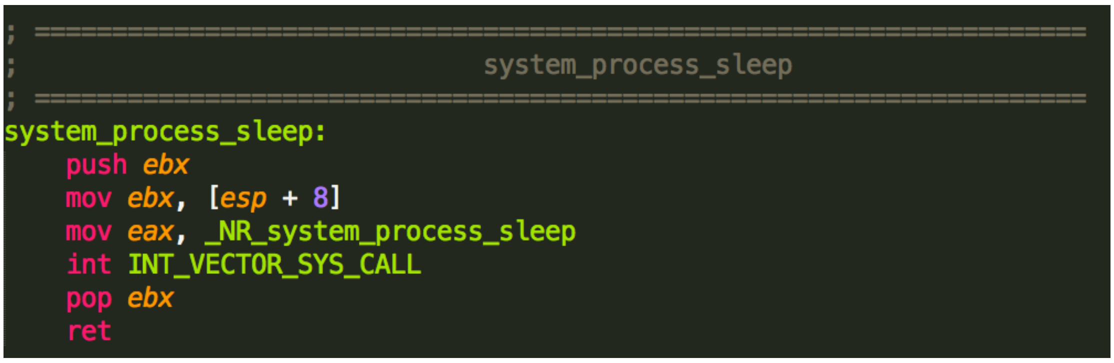
（1） 添加系统调用：

以sys_process_sleep为例，添加系统调用处理函数体 -> 在system.call中添加相应的用户调用函数体 -> 在proto.h中声明 system_process_sleep



```
/*======================================================================*
                           sys_process_sleep
 *======================================================================*/
PUBLIC int sys_process_sleep(int milli_sec){
    p_proc_ready -> sleep_ticks = milli_sec / 10;
    schedule();

    return 0;
}
```



```
; ====================================================================
;                              system_process_sleep
; ====================================================================
system_process_sleep:
    push ebx
    mov ebx, [esp + 8]
    mov eax, _NR_system_process_sleep
    int INT_VECTOR_SYS_CALL
    pop ebx
    ret
```

```
/* proc.c */
PUBLIC  int      sys_get_ticks();
PUBLIC int sys_process_sleep(int milli_sec);
PUBLIC  int      sys_disp_str(char* str);
PUBLIC int sys_P(SEMAPHORE* s);
PUBLIC int sys_V(SEMAPHORE* s);
```

其余同理，添加系统调 sys_disp_str时，还要在const.h中声明颜色数组，

并且在global.c中添加颜色数组

（2） 添加进程：

(参考orange’ s)

1.添加进程体

2.在task_table中增加一项(global.c)

3.让nr_tasks加1(proc.h)

4.定义任务堆栈(proc.h)

5.修改stack_size_total (proc.h)

```c
PUBLIC void barber(){
    while(1){
        system_disp_str("Barber is sleeping\n");

        system_P(first_semaphore + CUSTOMERS);
        system_P(first_semaphore + MUTEX);
        waiting--;
        system_V(first_semaphore + BARBER);
        system_V(first_semaphore + MUTEX);

        system_process_sleep(20000);
        system_disp_str("Barber finished cutting for ");
        itoa(buffer, now);
        system_disp_str(buffer);
        system_disp_str(" customer\n");
    }
}
```

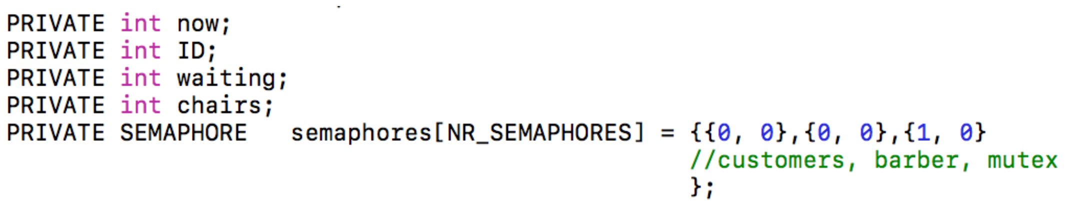
```c
PUBLIC  TASK    task_table[NR_TASKS] = {{task_tty, STACK_SIZE_TTY, "tty"},
                                        {Barber, STACK_SIZE_Barber, "Barber"},
                                        {Customer_A, STACK_SIZE_Customer_A, "Customer_A"},
                                        {Customer_B, STACK_SIZE_Customer_B, "Customer_B"},
                                        {Customer_C, STACK_SIZE_Customer_C, "Customer_C"}
                                    };
```

```c
/* Number of tasks */
#define NR_TASKS    5

/* stacks of tasks */
#define STACK_SIZE_TTY        0x8000
#define STACK_SIZE_Barber     0x8000
#define STACK_SIZE_Customer_A    0x8000
#define STACK_SIZE_Customer_B    0x8000
#define STACK_SIZE_Customer_C    0x8000

#define STACK_SIZE_TOTAL     (STACK_SIZE_TTY + \
                STACK_SIZE_Barber + \
                STACK_SIZE_Customer_A + \
                STACK_SIZE_Customer_B + \
                STACK_SIZE_Customer_C)
```

（3） 关于理发师问题：先声明信号量及有关变量:



```
PRIVATE int now;
PRIVATE int ID;
PRIVATE int waiting;
PRIVATE int chairs;
PRIVATE SEMAPHORE   semaphores[NR_SEMAPHORES] = {{0, 0},{0, 0},{1, 0}
                                                 //customers, barber, mutex
                                                 };
```

其中，now用来标记当前ID，ID用来记录总的ID，waiting记录正在等待的人，chairs用来记录椅子数，再参考书上的代码，加上一些输出语句完成该功能。

（4） 关于输出：

在第三次大作业中实现过的out_char的基础上，给out_char函数添加　个颜色参数，用来实现变色。