操作系统实验四

吴超月 131250168

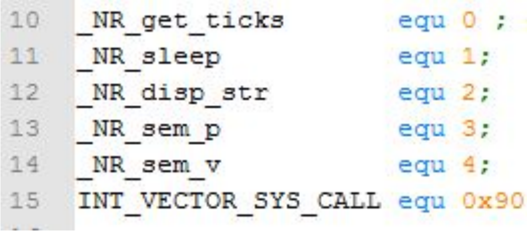
注：本代码是在 oranges 的第六章结合部分第七章代码的基础上完成的。

一、增加系统调用

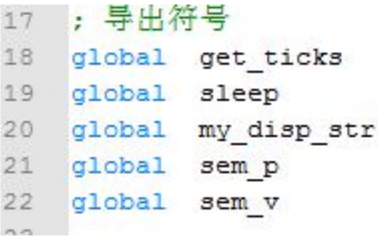
在原有代码的基础上增加四个作业要求的系统调用，步骤如下：

1、修改 syscall.asm

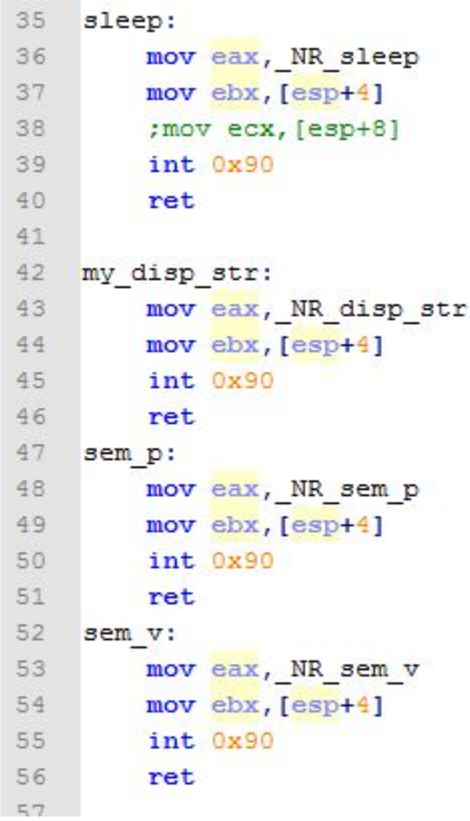
增加系统调用函数声明和定义：



```
_NR_get_ticks       equ 0 ;
_NR_sleep           equ 1;
_NR_disp_str        equ 2;
_NR_sem_p           equ 3;
_NR_sem_v           equ 4;
INT_VECTOR_SYS_CALL equ 0x90
```



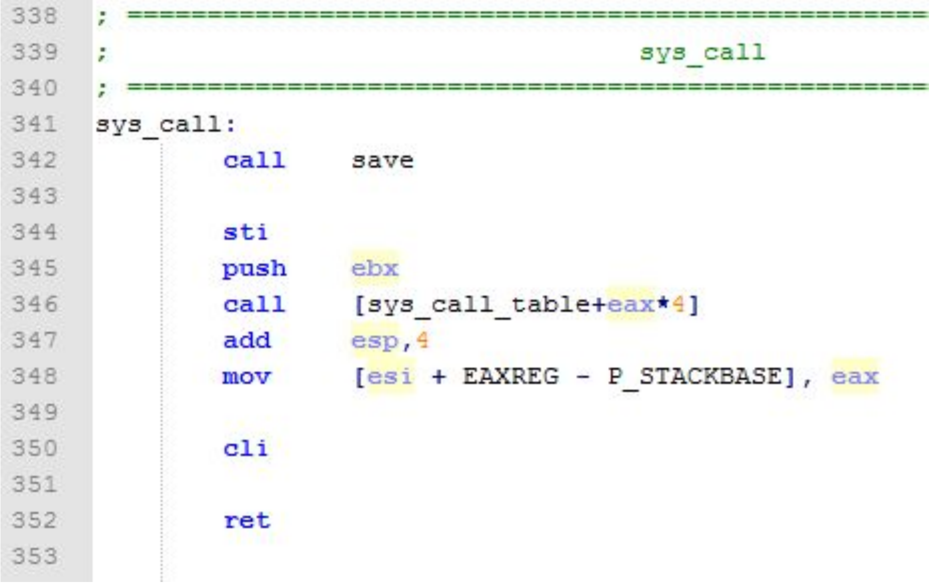
```
; 导出符号
global  get_ticks
global  sleep
global  my_disp_str
global  sem_p
global  sem_v
```



```
sleep:
    mov eax,_NR_sleep
    mov ebx,[esp+4]
    ;mov ecx,[esp+8]
    int 0x90
    ret

my_disp_str:
    mov eax,_NR_disp_str
    mov ebx,[esp+4]
    int 0x90
    ret
sem_p:
    mov eax,_NR_sem_p
    mov ebx,[esp+4]
    int 0x90
    ret
sem_v:
    mov eax,_NR_sem_v
    mov ebx,[esp+4]
    int 0x90
    ret
```

2、修改 kernel.asm

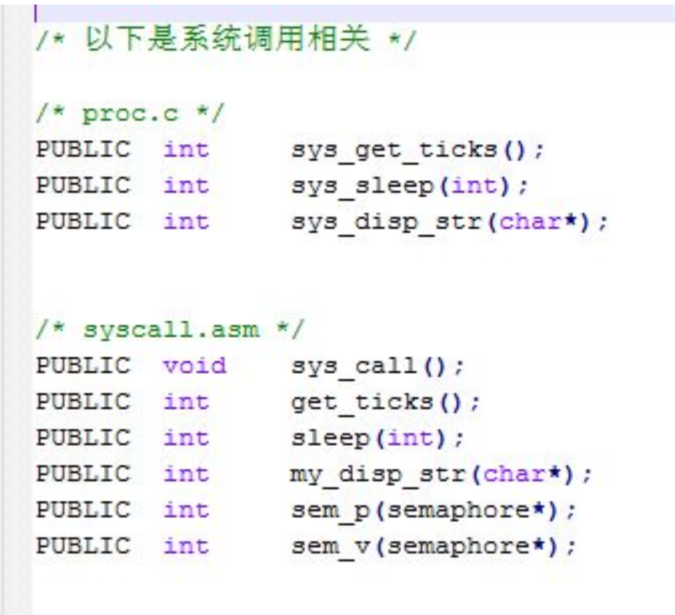
在 sys_call 中把参数压栈：



```
; ====================================================
;                                   sys_call
; ====================================================
sys_call:
        call    save

        sti
        push    ebx
        call    [sys_call_table+eax*4]
        add     esp,4
        mov     [esi + EAXREG - P_STACKBASE], eax

        cli

        ret

```

3、修改 proto.h

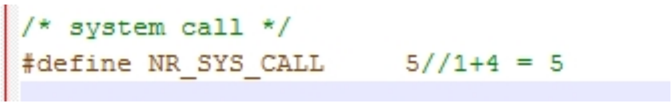
增加对四个函数的声明：



```
/* 以下是系统调用相关 */

/* proc.c */
PUBLIC  int     sys_get_ticks();
PUBLIC  int     sys_sleep(int);
PUBLIC  int     sys_disp_str(char*);


/* syscall.asm */
PUBLIC  void    sys_call();
PUBLIC  int     get_ticks();
PUBLIC  int     sleep(int);
PUBLIC  int     my_disp_str(char*);
PUBLIC  int     sem_p(semaphore*);
PUBLIC  int     sem_v(semaphore*);
```

4、修改 const.h

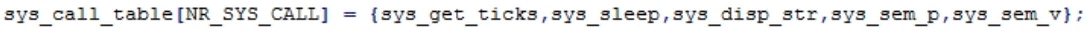
把系统调用的函数加四：



```
/* system call */
#define NR_SYS_CALL     5//1+4 = 5
```

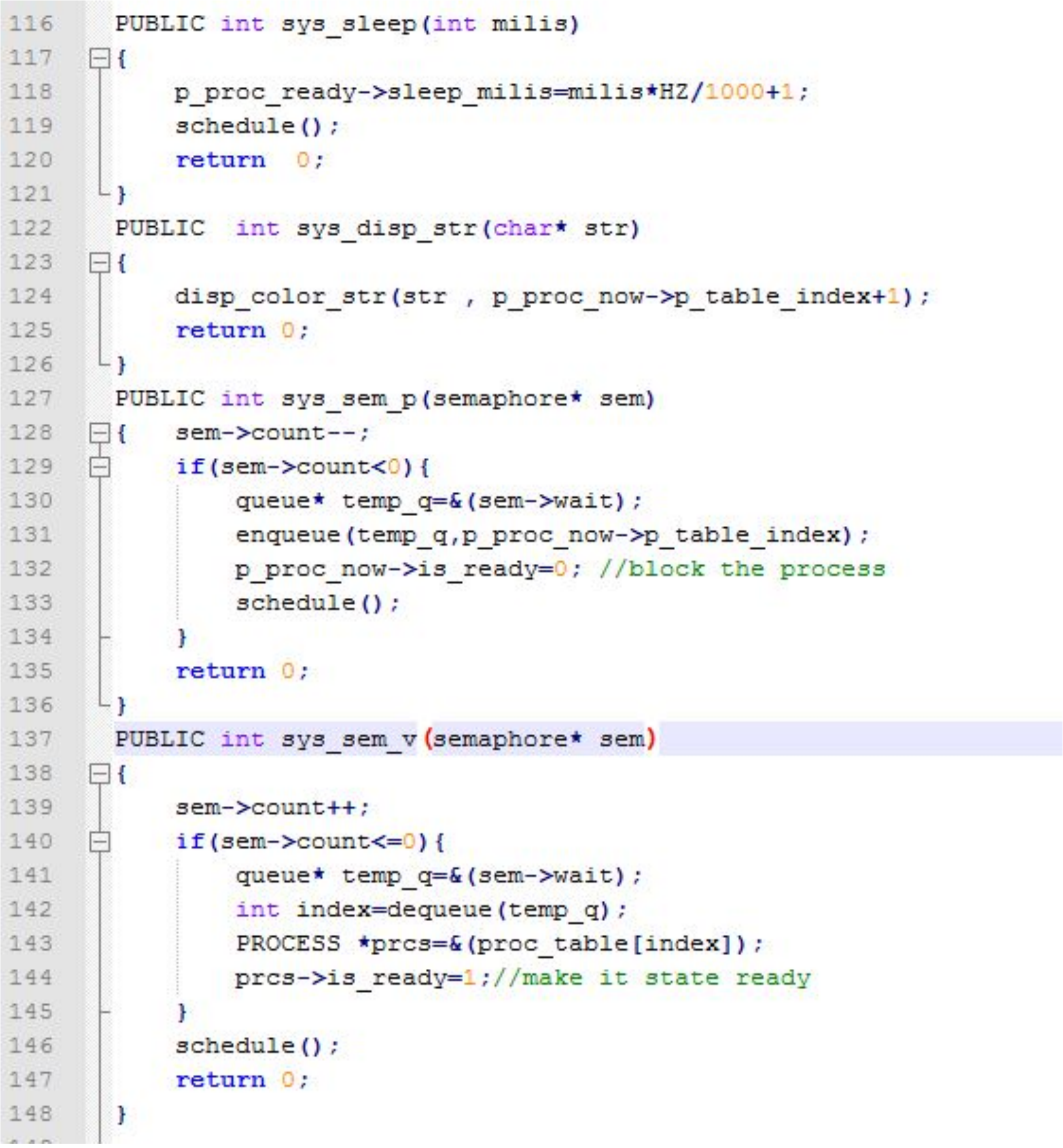
5、修改 global.c：

在 sys_call_table[]中对应添加系统调用



```
sys_call_table[NR_SYS_CALL] = {sys_get_ticks,sys_sleep,sys_disp_str,sys_sem_p,sys_sem_v};
```

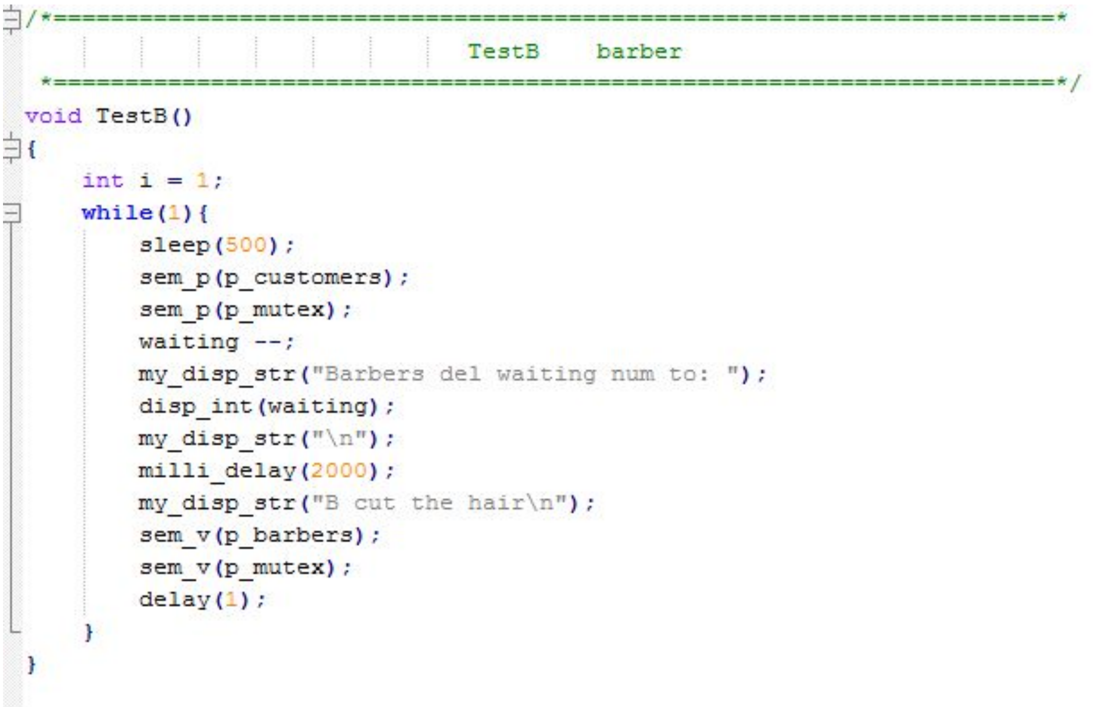
6、在 proc.c 中添加对四个系统调用的实现。



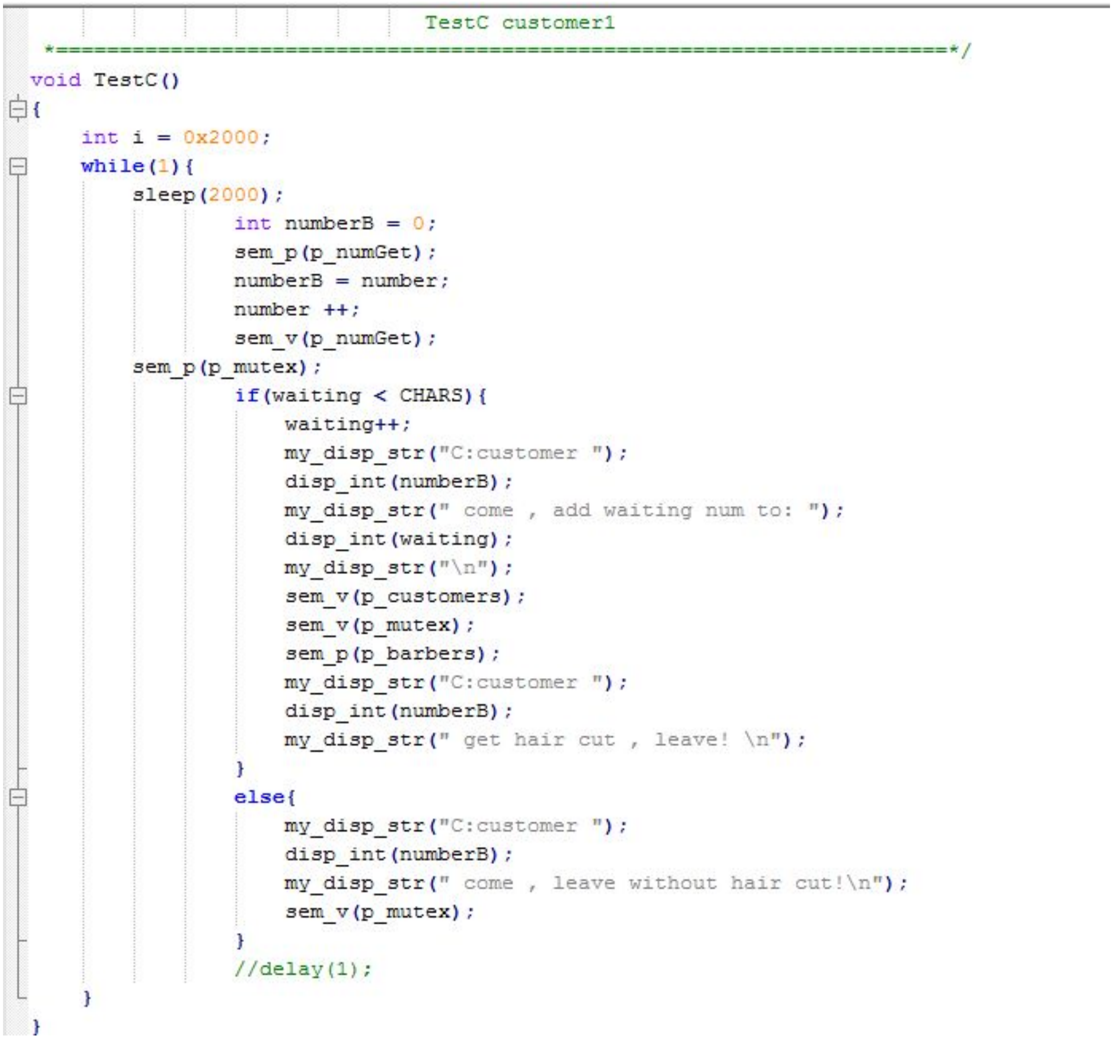
```
PUBLIC int sys_sleep(int milis)
{
    p_proc_ready->sleep_milis=milis*HZ/1000+1;
    schedule();
    return  0;
}
PUBLIC  int sys_disp_str(char* str)
{
    disp_color_str(str , p_proc_now->p_table_index+1);
    return 0;
}
PUBLIC int sys_sem_p(semaphore* sem)
{   sem->count--;
    if(sem->count<0){
        queue* temp_q=&(sem->wait);
        enqueue(temp_q,p_proc_now->p_table_index);
        p_proc_now->is_ready=0; //block the process
        schedule();
    }
    return 0;
}
PUBLIC int sys_sem_v (semaphore* sem)
{
    sem->count++;
    if(sem->count<=0){
        queue* temp_q=&(sem->wait);
        int index=dequeue(temp_q);
        PROCESS *prcs=&(proc_table[index]);
        prcs->is_ready=1;//make it state ready
    }
    schedule();
    return 0;
}
```

二、添加用户进程

1、在 main.c 中增加所用的进程



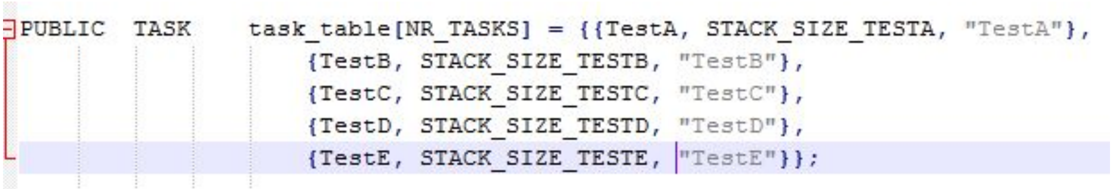
```
/*==========================================================================*
                                    TestB    barber
 *==========================================================================*/
void TestB()
{
    int i = 1;
    while(1){
        sleep(500);
        sem_p(p_customers);
        sem_p(p_mutex);
        waiting --;
        my_disp_str("Barbers del waiting num to: ");
        disp_int(waiting);
        my_disp_str("\n");
        milli_delay(2000);
        my_disp_str("B cut the hair\n");
        sem_v(p_barbers);
        sem_v(p_mutex);
        delay(1);
    }
}
```



```
                                    TestC customer1
 *==================================================================*/
void TestC()
{
    int i = 0x2000;
    while(1){
        sleep(2000);
                int numberB = 0;
                sem_p(p_numGet);
                numberB = number;
                number ++;
                sem_v(p_numGet);
        sem_p(p_mutex);
                if(waiting < CHARS){
                    waiting++;
                    my_disp_str("C:customer ");
                    disp_int(numberB);
                    my_disp_str(" come , add waiting num to: ");
                    disp_int(waiting);
                    my_disp_str("\n");
                    sem_v(p_customers);
                    sem_v(p_mutex);
                    sem_p(p_barbers);
                    my_disp_str("C:customer ");
                    disp_int(numberB);
                    my_disp_str(" get hair cut , leave! \n");
                }
                else{
                    my_disp_str("C:customer ");
                    disp_int(numberB);
                    my_disp_str(" come , leave without hair cut!\n");
                    sem_v(p_mutex);
                }
                //delay(1);
    }
}
```

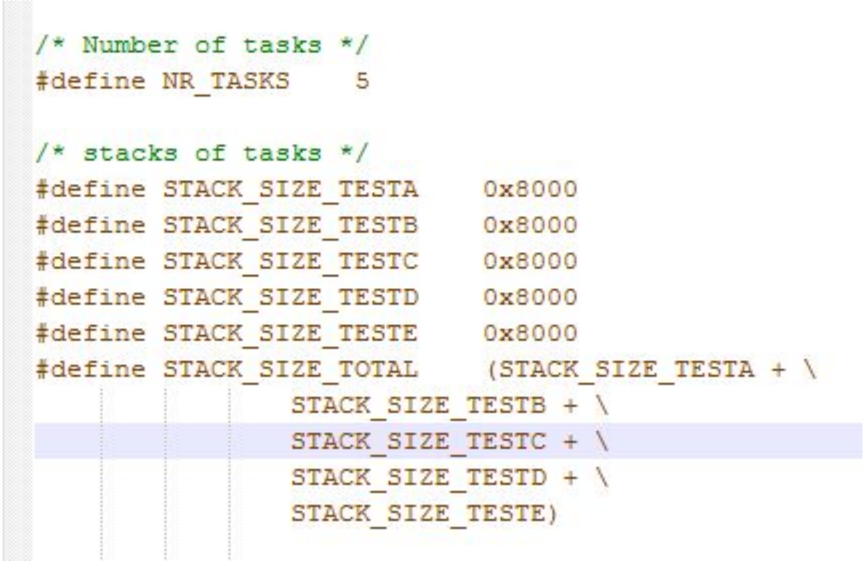
（余下两个进程同理）

2、在 global.c 的 task_table 中新增所用进程



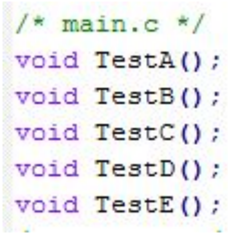
```
PUBLIC  TASK    task_table[NR_TASKS] = {{TestA, STACK_SIZE_TESTA, "TestA"},
                    {TestB, STACK_SIZE_TESTB, "TestB"},
                    {TestC, STACK_SIZE_TESTC, "TestC"},
                    {TestD, STACK_SIZE_TESTD, "TestD"},
                    {TestE, STACK_SIZE_TESTE, "TestE"}};
```

3、修改在 proc.h 的 NR_TASKS 的值，并给新增加的进程定义进程栈，修改栈的总大小



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

/* stacks of tasks */
#define STACK_SIZE_TESTA    0x8000
#define STACK_SIZE_TESTB    0x8000
#define STACK_SIZE_TESTC    0x8000
#define STACK_SIZE_TESTD    0x8000
#define STACK_SIZE_TESTE    0x8000
#define STACK_SIZE_TOTAL    (STACK_SIZE_TESTA + \
                STACK_SIZE_TESTB + \
                STACK_SIZE_TESTC + \
                STACK_SIZE_TESTD + \
                STACK_SIZE_TESTE)
```

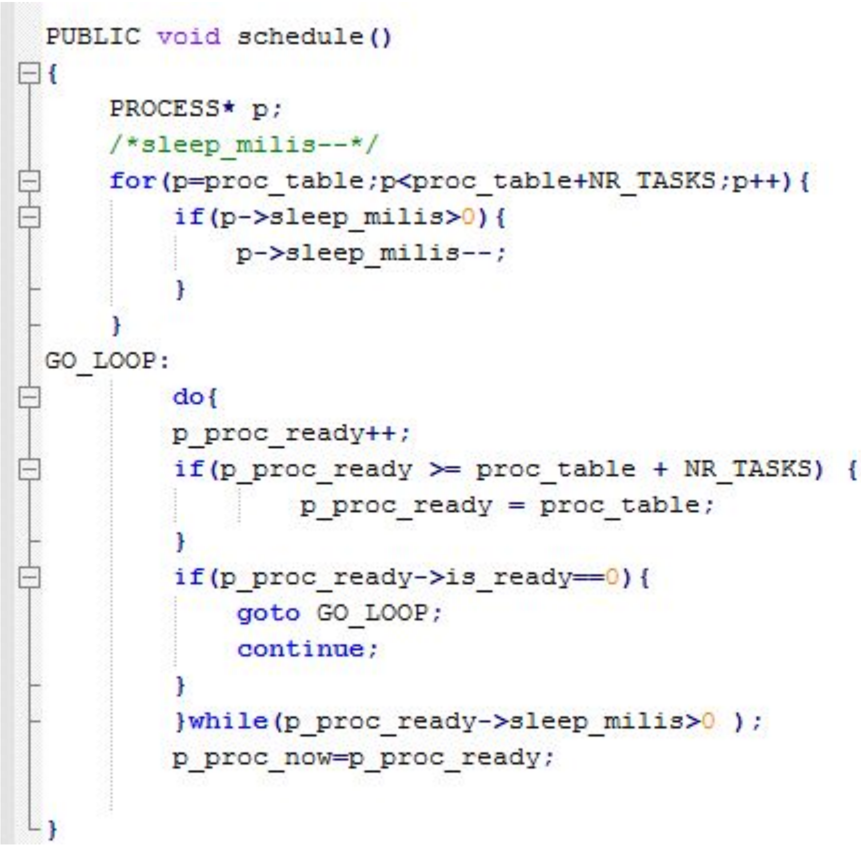
4、在 proto.h 中声明新增加的进程



```
/* main.c */
void TestA();
void TestB();
void TestC();
void TestD();
void TestE();
```

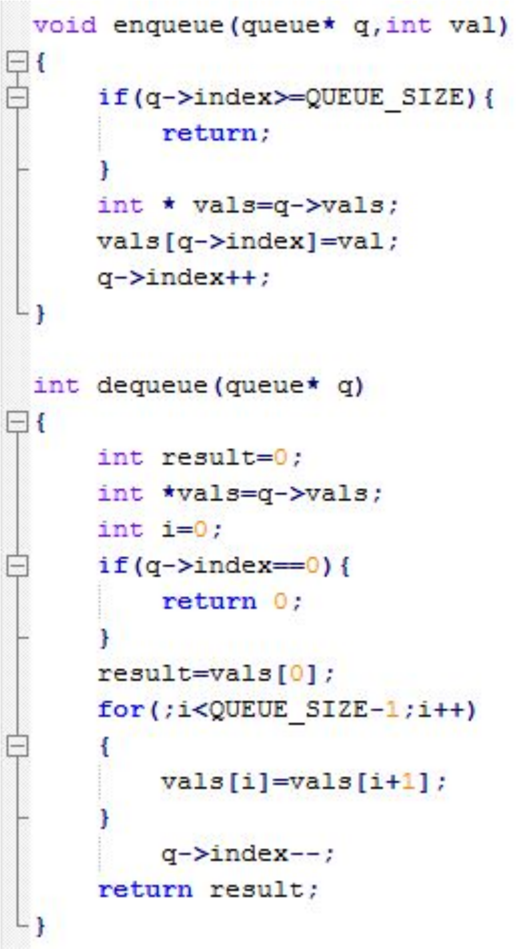
三、其他修改与增加

1、修改 proc.h 中的 schedule()

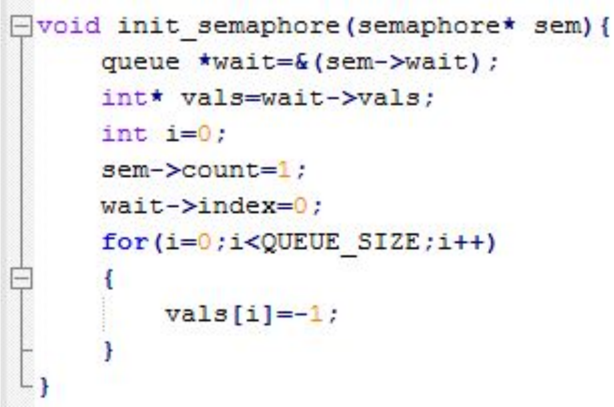


```
PUBLIC void schedule()
{
    PROCESS* p;
    /*sleep_milis--*/
    for(p=proc_table;p<proc_table+NR_TASKS;p++){
        if(p->sleep_milis>0){
            p->sleep_milis--;
        }
    }
GO_LOOP:
        do{
        p_proc_ready++;
        if(p_proc_ready >= proc_table + NR_TASKS) {
                p_proc_ready = proc_table;
        }
        if(p_proc_ready->is_ready==0){
            goto GO_LOOP;
            continue;
        }
        }while(p_proc_ready->sleep_milis>0 );
        p_proc_now=p_proc_ready;

}
```

2、定义供信号量操作所用的队列函数，并增加初始化信号量的函数：

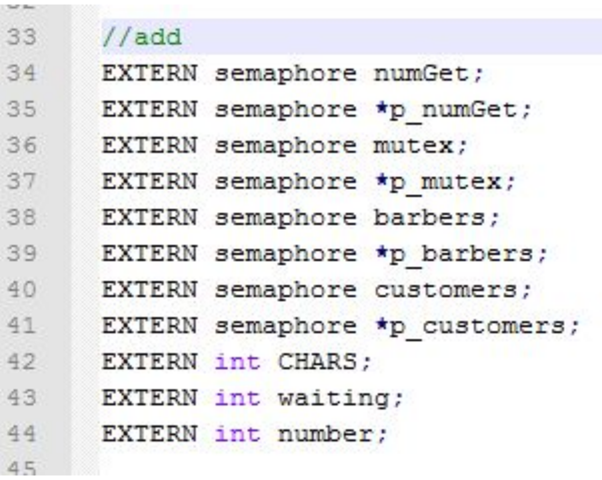


```
void enqueue(queue* q,int val)
{
    if(q->index>=QUEUE_SIZE){
        return;
    }
    int * vals=q->vals;
    vals[q->index]=val;
    q->index++;
}

int dequeue(queue* q)
{
    int result=0;
    int *vals=q->vals;
    int i=0;
    if(q->index==0){
        return 0;
    }
    result=vals[0];
    for(;i<QUEUE_SIZE-1;i++)
    {
        vals[i]=vals[i+1];
    }
        q->index--;
    return result;
}
```



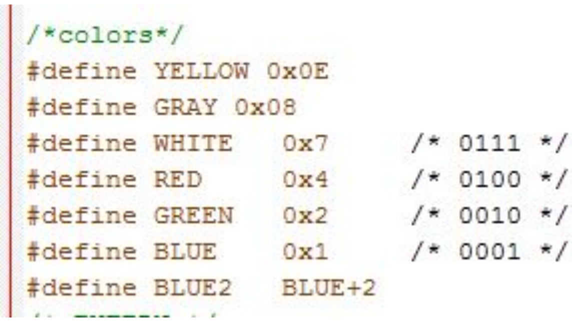
```
void init_semaphore(semaphore* sem){
    queue *wait=&(sem->wait);
    int* vals=wait->vals;
    int i=0;
    sem->count=1;
    wait->index=0;
    for(i=0;i<QUEUE_SIZE;i++)
    {
        vals[i]=-1;
    }
}
```

3、在 global.h 中增加进程调度所用的变量的声明：



```
//add
EXTERN semaphore numGet;
EXTERN semaphore *p_numGet;
EXTERN semaphore mutex;
EXTERN semaphore *p_mutex;
EXTERN semaphore barbers;
EXTERN semaphore *p_barbers;
EXTERN semaphore customers;
EXTERN semaphore *p_customers;
EXTERN int CHARS;
EXTERN int waiting;
EXTERN int number;
```

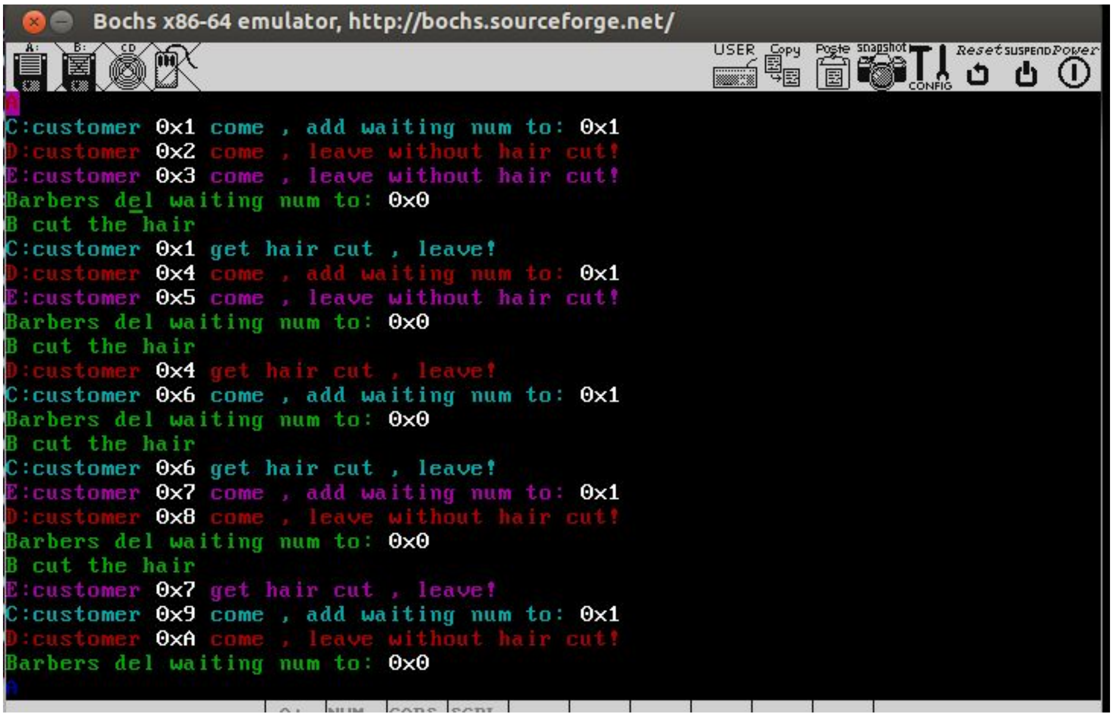
4、在 const.h 中增加对颜色宏的定义，用来区分打印出的内容



```
/*colors*/
#define YELLOW 0x0E
#define GRAY 0x08
#define WHITE   0x7     /* 0111 */
#define RED     0x4     /* 0100 */
#define GREEN   0x2     /* 0010 */
#define BLUE    0x1     /* 0001 */
#define BLUE2   BLUE+2
```

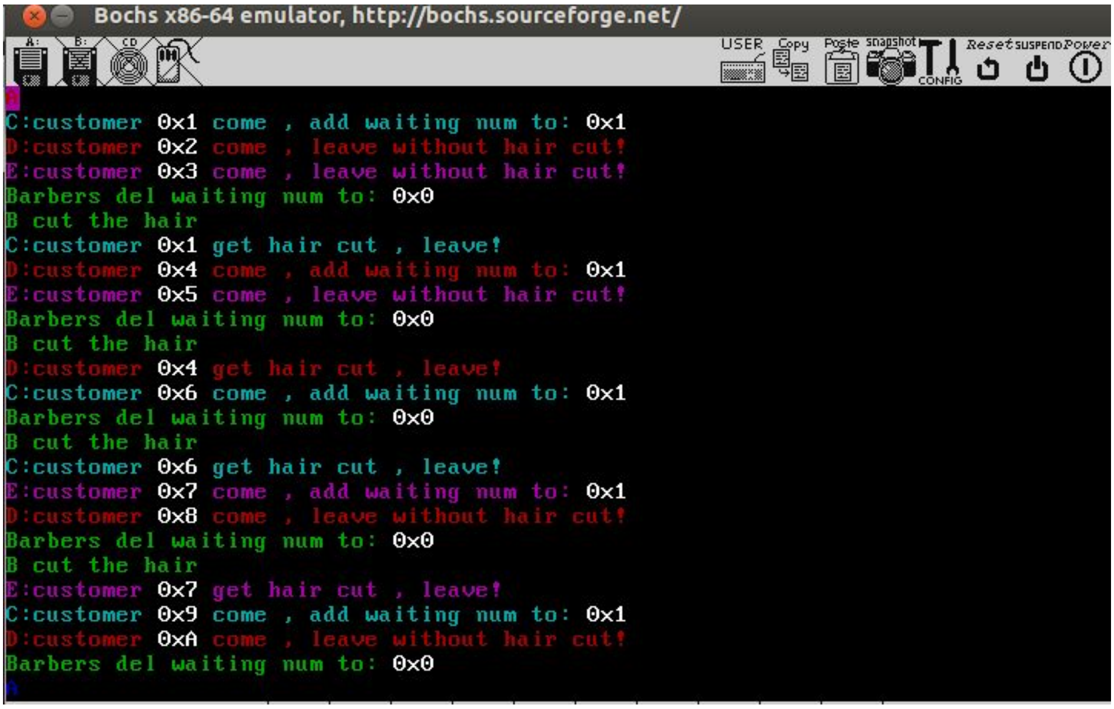
四、运行截图

chair = 1



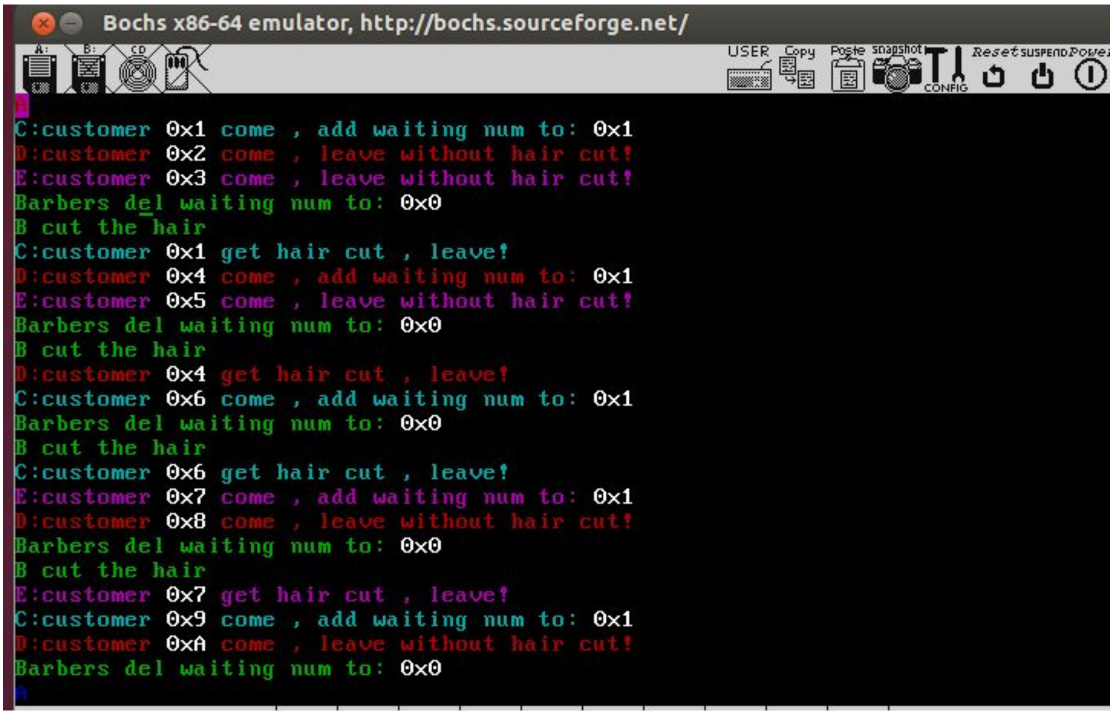
```
A
C:customer 0x1 come , add waiting num to: 0x1
D:customer 0x2 come , leave without hair cut!
E:customer 0x3 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
C:customer 0x1 get hair cut , leave!
D:customer 0x4 come , add waiting num to: 0x1
E:customer 0x5 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
D:customer 0x4 get hair cut , leave!
C:customer 0x6 come , add waiting num to: 0x1
Barbers del waiting num to: 0x0
B cut the hair
C:customer 0x6 get hair cut , leave!
E:customer 0x7 come , add waiting num to: 0x1
D:customer 0x8 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
E:customer 0x7 get hair cut , leave!
C:customer 0x9 come , add waiting num to: 0x1
D:customer 0xA come , leave without hair cut!
Barbers del waiting num to: 0x0
A
```

Chair= 2



```
A
C:customer 0x1 come , add waiting num to: 0x1
D:customer 0x2 come , leave without hair cut!
E:customer 0x3 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
C:customer 0x1 get hair cut , leave!
D:customer 0x4 come , add waiting num to: 0x1
E:customer 0x5 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
D:customer 0x4 get hair cut , leave!
C:customer 0x6 come , add waiting num to: 0x1
Barbers del waiting num to: 0x0
B cut the hair
C:customer 0x6 get hair cut , leave!
E:customer 0x7 come , add waiting num to: 0x1
D:customer 0x8 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
E:customer 0x7 get hair cut , leave!
C:customer 0x9 come , add waiting num to: 0x1
D:customer 0xA come , leave without hair cut!
Barbers del waiting num to: 0x0
A
```

Chair = 3



```
A
C:customer 0x1 come , add waiting num to: 0x1
D:customer 0x2 come , leave without hair cut!
E:customer 0x3 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
C:customer 0x1 get hair cut , leave!
D:customer 0x4 come , add waiting num to: 0x1
E:customer 0x5 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
D:customer 0x4 get hair cut , leave!
C:customer 0x6 come , add waiting num to: 0x1
Barbers del waiting num to: 0x0
B cut the hair
C:customer 0x6 get hair cut , leave!
E:customer 0x7 come , add waiting num to: 0x1
D:customer 0x8 come , leave without hair cut!
Barbers del waiting num to: 0x0
B cut the hair
E:customer 0x7 get hair cut , leave!
C:customer 0x9 come , add waiting num to: 0x1
D:customer 0xA come , leave without hair cut!
Barbers del waiting num to: 0x0
A
```