实验四修改记录

151250132 唐鑫

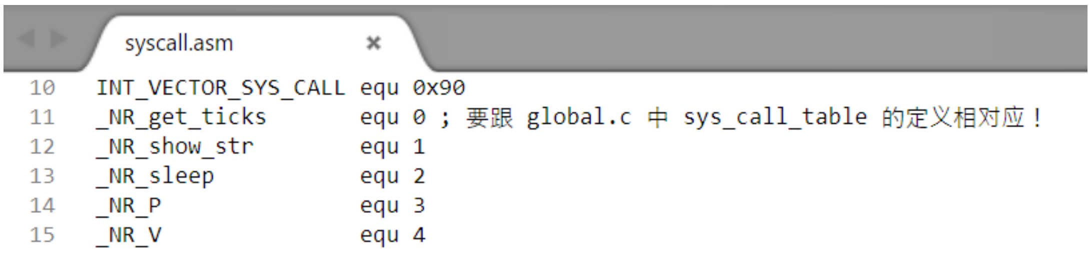
Makefile:
改 make image 为 make run，
CFLAGS 中加入"-fno-stack-protector"(否则编译会遇到“__stack_chk_fail_local”错误)

syscall.asm:增加系统调用函数声明和定义：

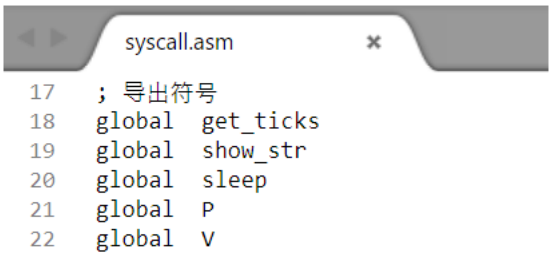
show_str: 添加一个系统调用sys_disp_str，其功能接受一个char* str参数，打印出字符串

sleep: 添加一个系统调用 sys_process_sleep，其功能是接受一个 int 型参数 mill_seconds，调用此 System Call 的进程会在 mill_seconds 毫秒内不被进程调度函数分配时间片

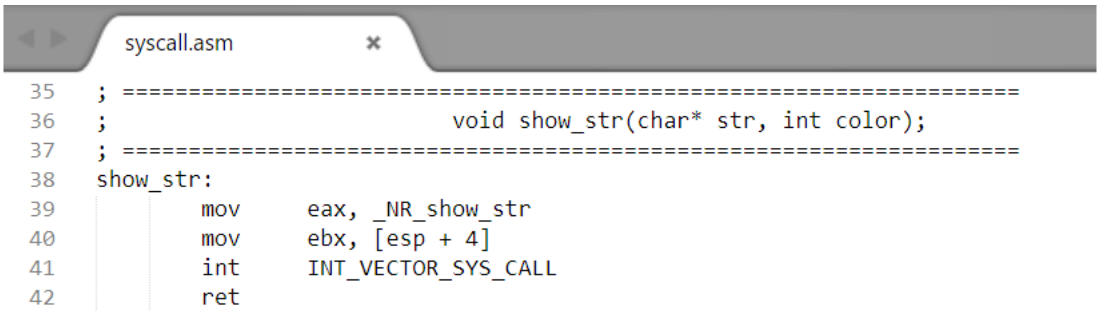
P/V: 添加两个系统调用sys_sem_p和sys_sem_v，即信号量的PV操作，在此基础上模拟睡眠的理发师问题



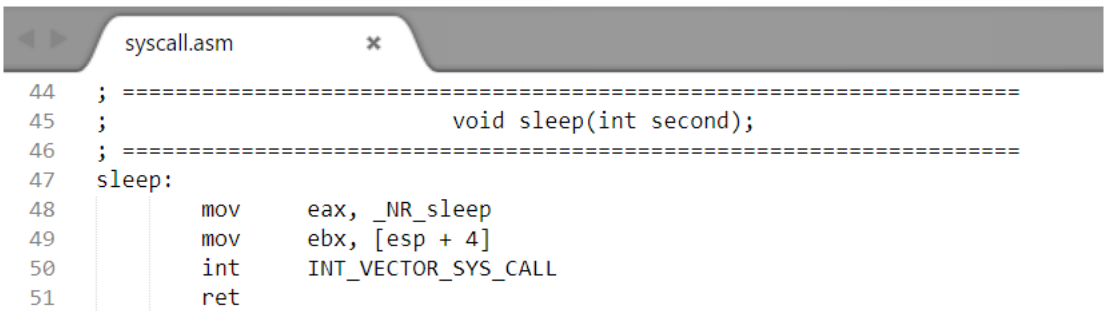
```
INT_VECTOR_SYS_CALL equ 0x90
_NR_get_ticks       equ 0 ; 要跟 global.c 中 sys_call_table 的定义相对应！
_NR_show_str        equ 1
_NR_sleep           equ 2
_NR_P               equ 3
_NR_V               equ 4
```



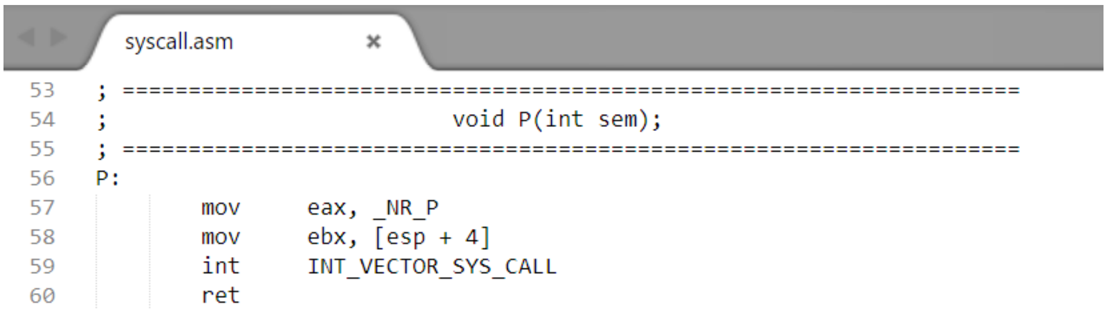
```
; 导出符号
global  get_ticks
global  show_str
global  sleep
global  P
global  V
```



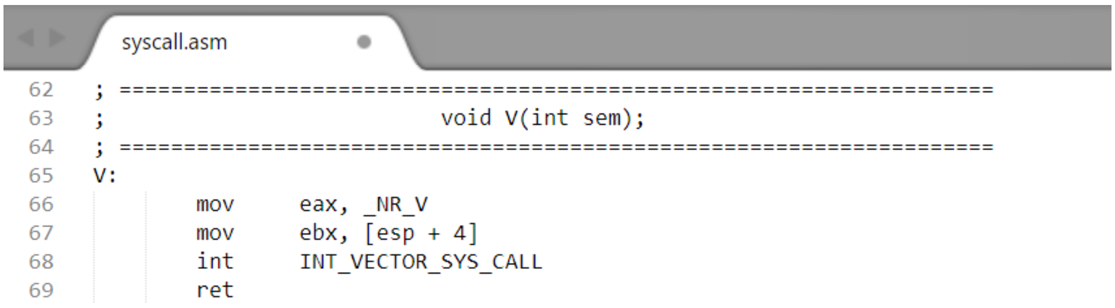
```
; ====================================================================
;                          void show_str(char* str, int color);
; ====================================================================
show_str:
        mov     eax, _NR_show_str
        mov     ebx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret
```



```
; ====================================================================
;                          void sleep(int second);
; ====================================================================
sleep:
        mov     eax, _NR_sleep
        mov     ebx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret
```

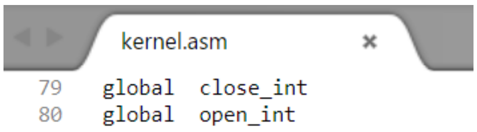


```
; ======================================================================
;                          void P(int sem);
; ======================================================================
P:
        mov     eax, _NR_P
        mov     ebx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret
```

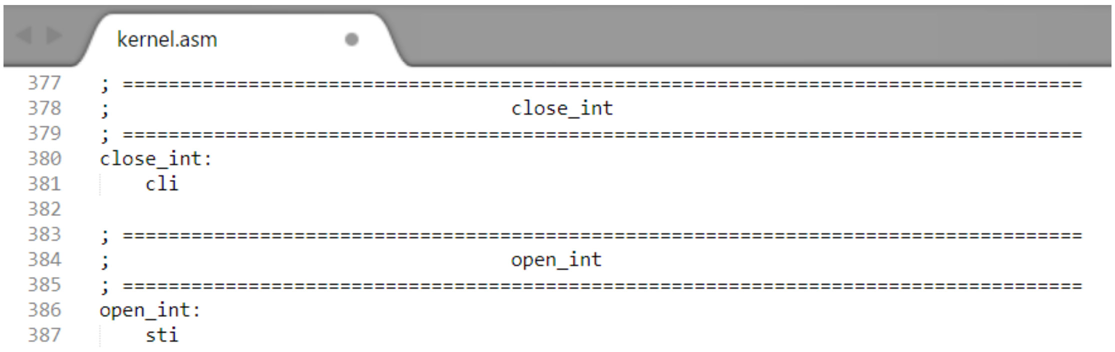


```
; ======================================================================
;                          void V(int sem);
; ======================================================================
V:
        mov     eax, _NR_V
        mov     ebx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret
```

kernel.asm:开关中断

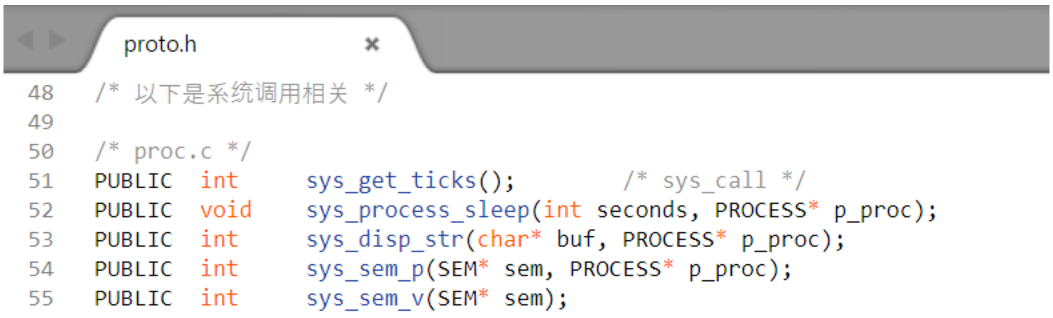


```
global  close_int
global  open_int
```

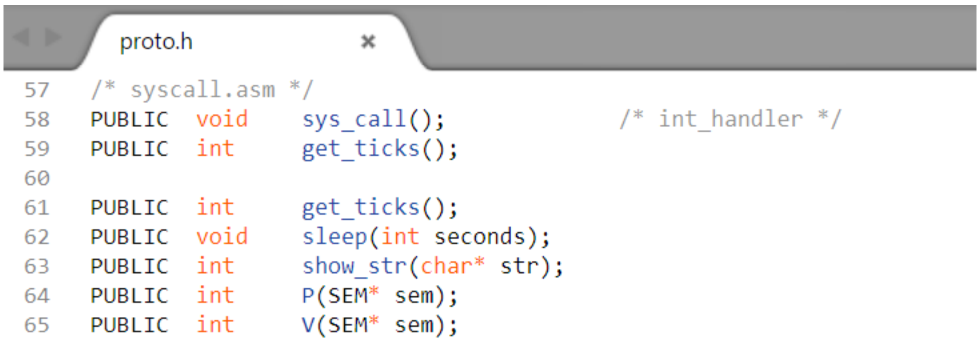


```
; ======================================================================
;                              close_int
; ======================================================================
close_int:
    cli

; ======================================================================
;                              open_int
; ======================================================================
open_int:
    sti
```

proto.h:添加对四个函数的声明

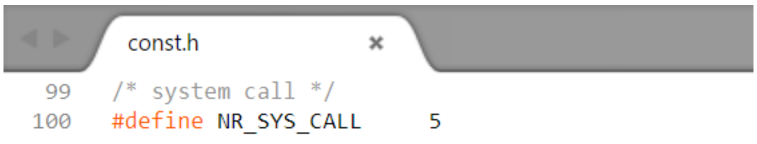


```
/* 以下是系统调用相关 */

/* proc.c */
PUBLIC  int     sys_get_ticks();         /* sys_call */
PUBLIC  void    sys_process_sleep(int seconds, PROCESS* p_proc);
PUBLIC  int     sys_disp_str(char* buf, PROCESS* p_proc);
PUBLIC  int     sys_sem_p(SEM* sem, PROCESS* p_proc);
PUBLIC  int     sys_sem_v(SEM* sem);
```



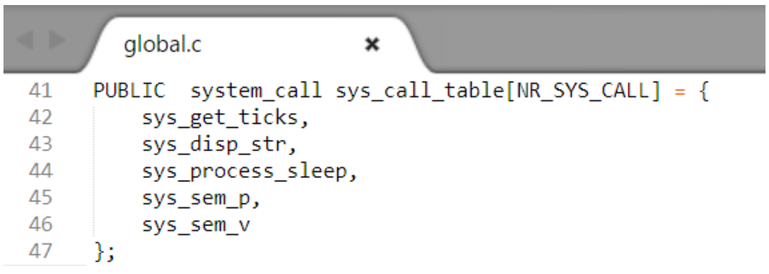
```
/* syscall.asm */
PUBLIC  void    sys_call();             /* int_handler */
PUBLIC  int     get_ticks();

PUBLIC  int     get_ticks();
PUBLIC  void    sleep(int seconds);
PUBLIC  int     show_str(char* str);
PUBLIC  int     P(SEM* sem);
PUBLIC  int     V(SEM* sem);
```

const.h:将系统调用函数数目改为 5



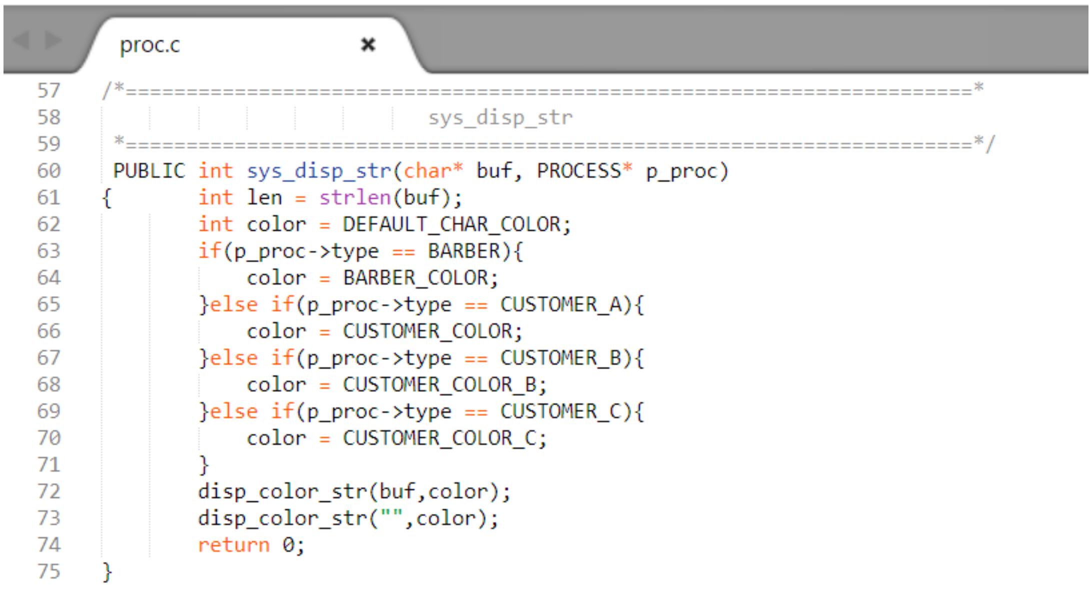
```
/* system call */
#define NR_SYS_CALL     5
```

global.c:在 sys_call_table 中添加四个系统调用

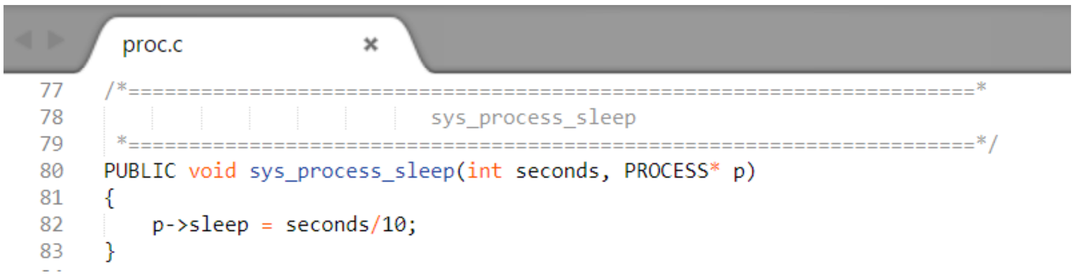


```
PUBLIC  system_call sys_call_table[NR_SYS_CALL] = {
    sys_get_ticks,
    sys_disp_str,
    sys_process_sleep,
    sys_sem_p,
    sys_sem_v
};
```

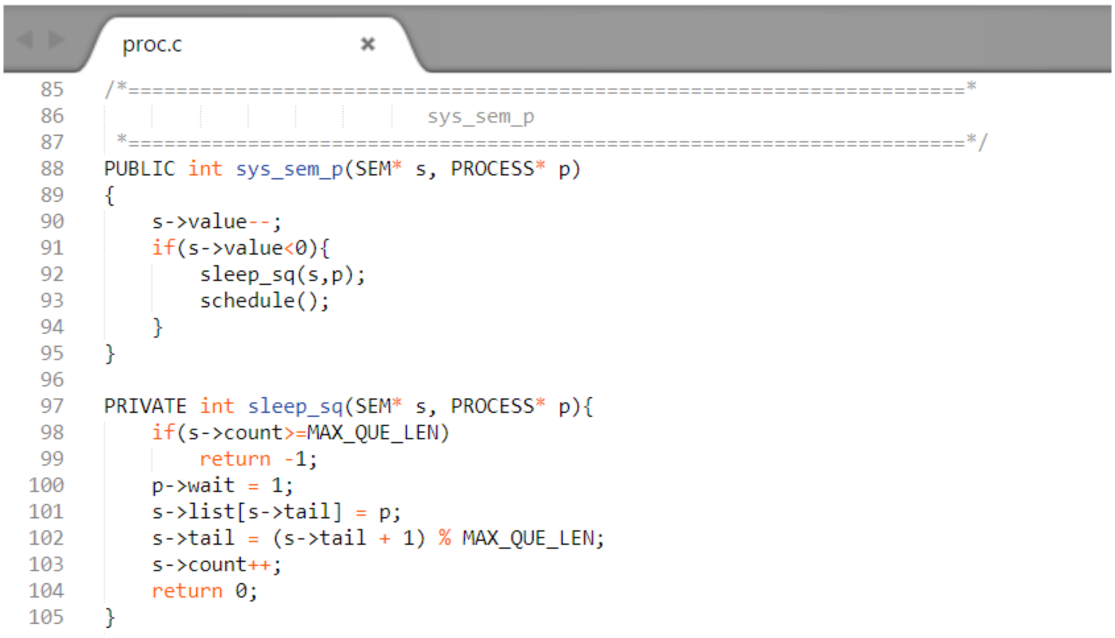
proc.c:添加四个系统调用的实现



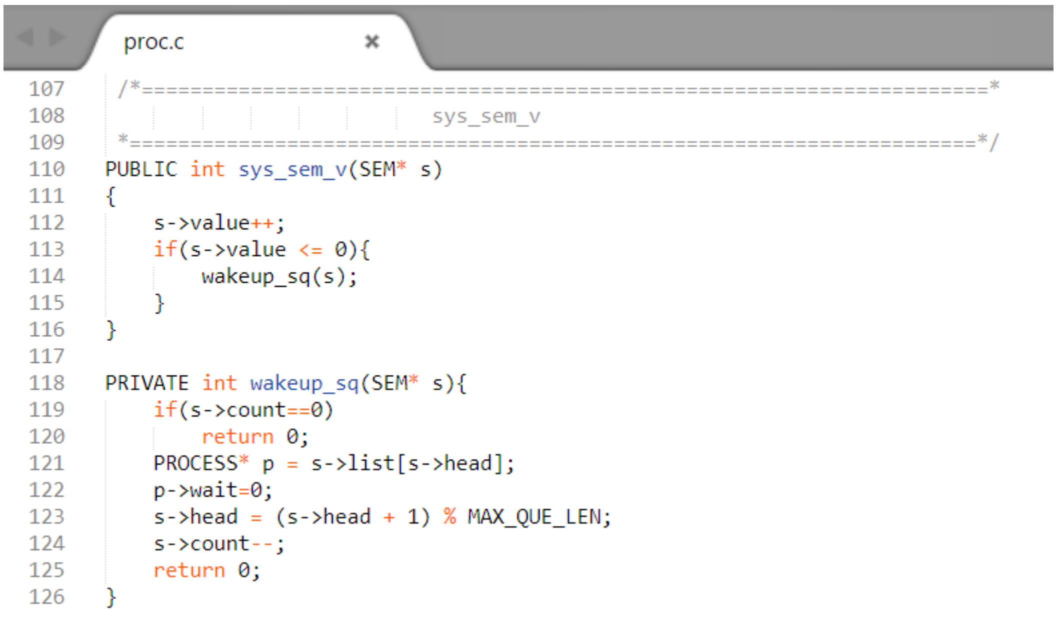
```
/*======================================================================*
                        sys_disp_str
 *======================================================================*/
 PUBLIC int sys_disp_str(char* buf, PROCESS* p_proc)
{       int len = strlen(buf);
        int color = DEFAULT_CHAR_COLOR;
        if(p_proc->type == BARBER){
            color = BARBER_COLOR;
        }else if(p_proc->type == CUSTOMER_A){
            color = CUSTOMER_COLOR;
        }else if(p_proc->type == CUSTOMER_B){
            color = CUSTOMER_COLOR_B;
        }else if(p_proc->type == CUSTOMER_C){
            color = CUSTOMER_COLOR_C;
        }
        disp_color_str(buf,color);
        disp_color_str("",color);
        return 0;
}
```



```c
/*======================================================================*
                          sys_process_sleep
 *======================================================================*/
PUBLIC void sys_process_sleep(int seconds, PROCESS* p)
{
    p->sleep = seconds/10;
}
```

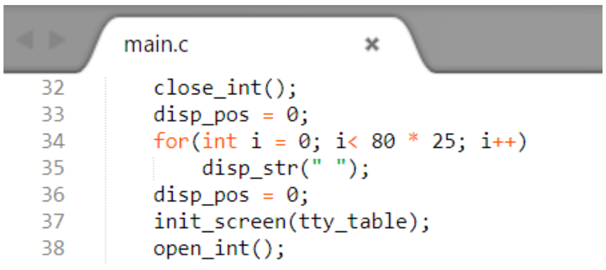


```c
/*======================================================================*
                          sys_sem_p
 *======================================================================*/
PUBLIC int sys_sem_p(SEM* s, PROCESS* p)
{
    s->value--;
    if(s->value<0){
        sleep_sq(s,p);
        schedule();
    }
}

PRIVATE int sleep_sq(SEM* s, PROCESS* p){
    if(s->count>=MAX_QUE_LEN)
        return -1;
    p->wait = 1;
    s->list[s->tail] = p;
    s->tail = (s->tail + 1) % MAX_QUE_LEN;
    s->count++;
    return 0;
}
```

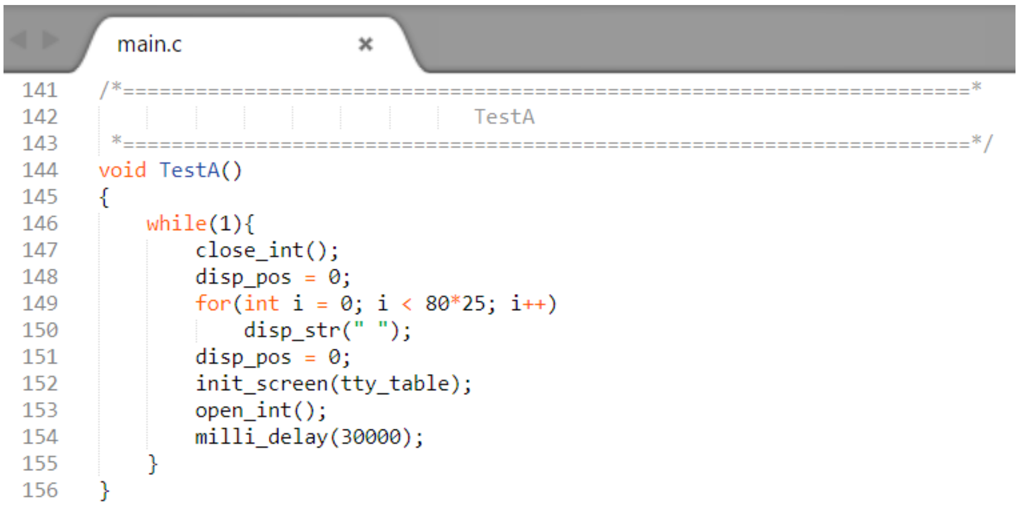


```c
/*======================================================================*
                          sys_sem_v
 *======================================================================*/
PUBLIC int sys_sem_v(SEM* s)
{
    s->value++;
    if(s->value <= 0){
        wakeup_sq(s);
    }
}

PRIVATE int wakeup_sq(SEM* s){
    if(s->count==0)
        return 0;
    PROCESS* p = s->list[s->head];
    p->wait=0;
    s->head = (s->head + 1) % MAX_QUE_LEN;
    s->count--;
    return 0;
}
```

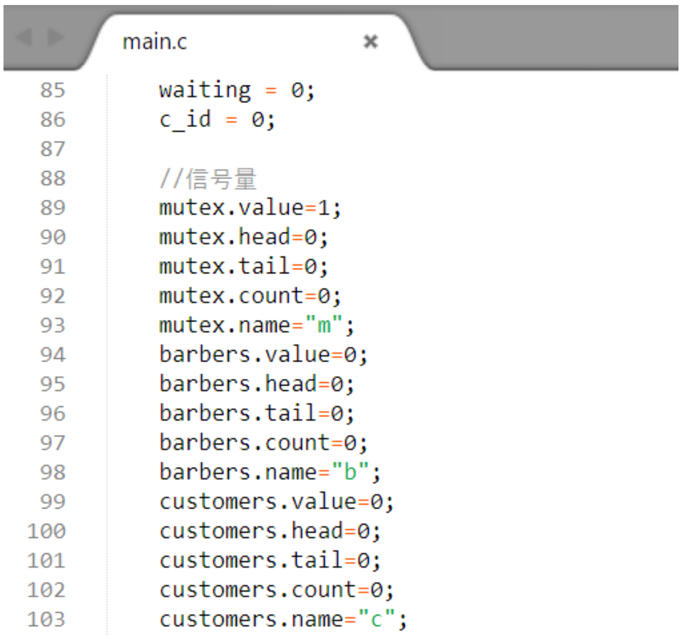
main.c:清屏操作；增加进程



```c
    close_int();
    disp_pos = 0;
    for(int i = 0; i< 80 * 25; i++)
        disp_str(" ");
    disp_pos = 0;
    init_screen(tty_table);
    open_int();
```

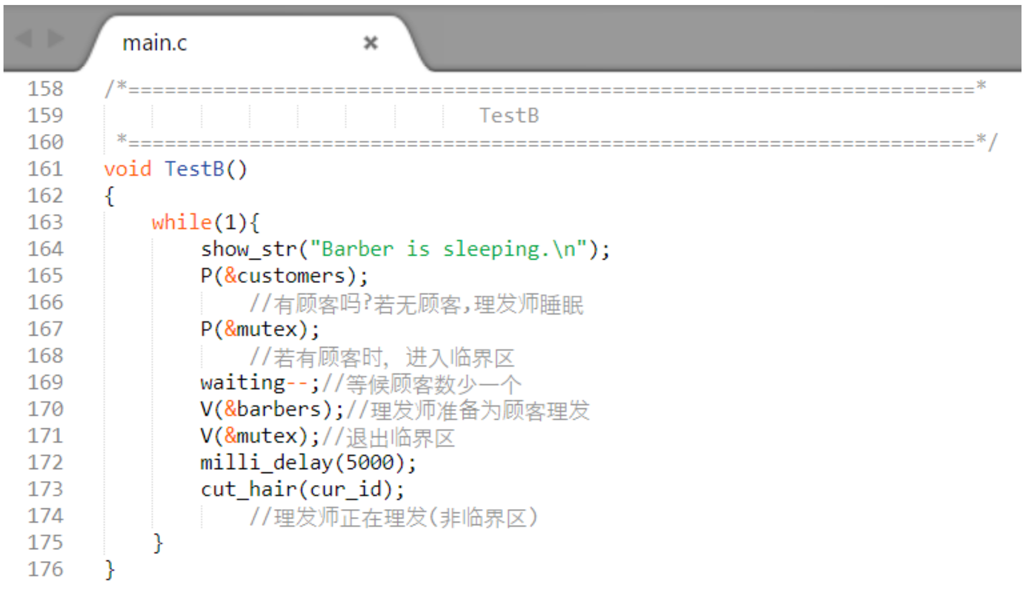


```c
/*======================================================================*
                                   TestA
 *======================================================================*/
void TestA()
{
    while(1){
        close_int();
        disp_pos = 0;
        for(int i = 0; i < 80*25; i++)
            disp_str(" ");
        disp_pos = 0;
        init_screen(tty_table);
        open_int();
        milli_delay(30000);
    }
}
```

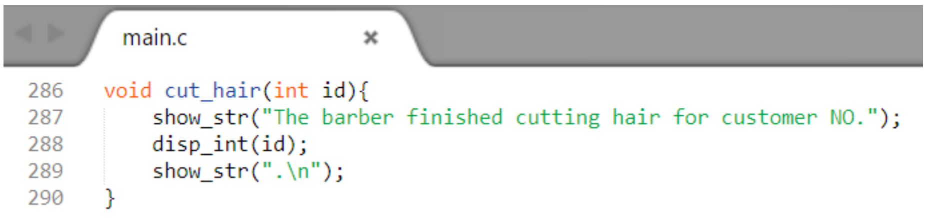


```c
    waiting = 0;
    c_id = 0;

    //信号量
    mutex.value=1;
    mutex.head=0;
    mutex.tail=0;
    mutex.count=0;
    mutex.name="m";
    barbers.value=0;
    barbers.head=0;
    barbers.tail=0;
    barbers.count=0;
    barbers.name="b";
    customers.value=0;
    customers.head=0;
    customers.tail=0;
    customers.count=0;
    customers.name="c";
```

Barber 理发师进程：

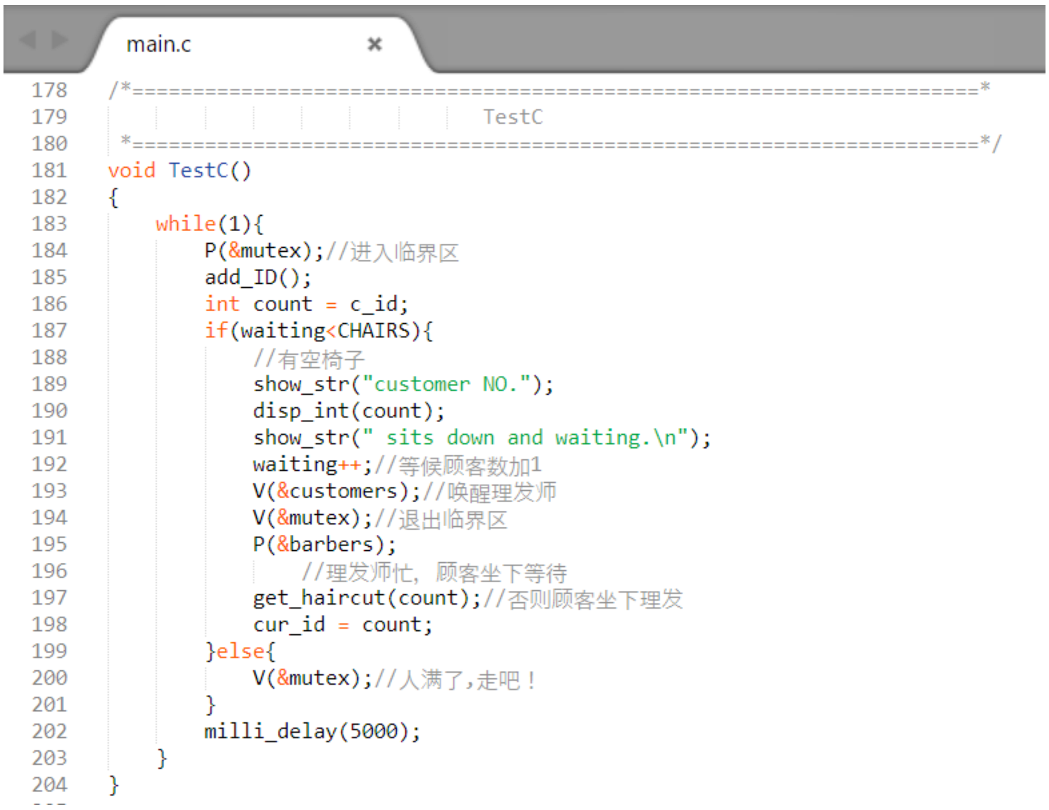

main.c

```c
/*=====================================================================*
                          TestB
 *=====================================================================*/
void TestB()
{
    while(1){
        show_str("Barber is sleeping.\n");
        P(&customers);
            //有顾客吗?若无顾客,理发师睡眠
        P(&mutex);
            //若有顾客时, 进入临界区
        waiting--;//等候顾客数少一个
        V(&barbers);//理发师准备为顾客理发
        V(&mutex);//退出临界区
        milli_delay(5000);
        cut_hair(cur_id);
            //理发师正在理发(非临界区)
    }
}
```

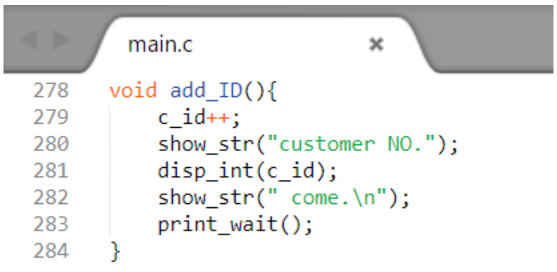

main.c

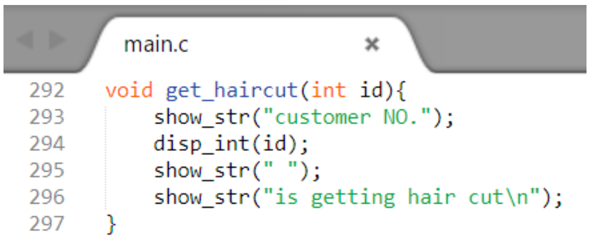
```c
void cut_hair(int id){
    show_str("The barber finished cutting hair for customer NO.");
    disp_int(id);
    show_str(".\n");
}
```

costomer 顾客进程 C(D、E 相同):

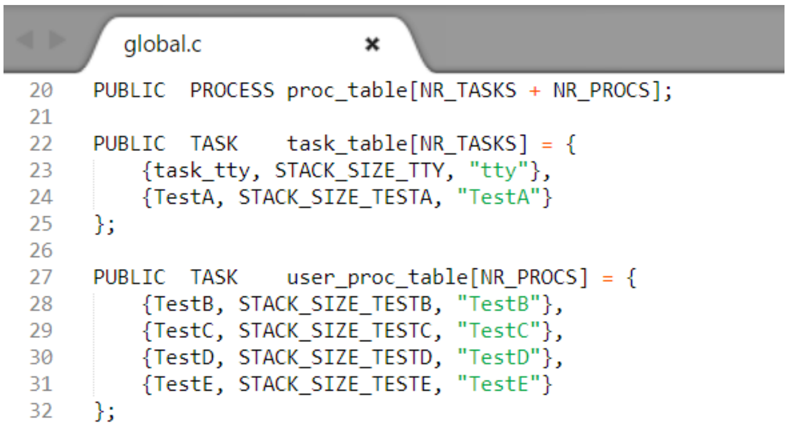

main.c

```c
/*=====================================================================*
                          TestC
 *=====================================================================*/
void TestC()
{
    while(1){
        P(&mutex);//进入临界区
        add_ID();
        int count = c_id;
        if(waiting<CHAIRS){
            //有空椅子
            show_str("customer NO.");
            disp_int(count);
            show_str(" sits down and waiting.\n");
            waiting++;//等候顾客数加1
            V(&customers);//唤醒理发师
            V(&mutex);//退出临界区
            P(&barbers);
                //理发师忙, 顾客坐下等待
            get_haircut(count);//否则顾客坐下理发
            cur_id = count;
        }else{
            V(&mutex);//人满了,走吧！
        }
        milli_delay(5000);
    }
}
```

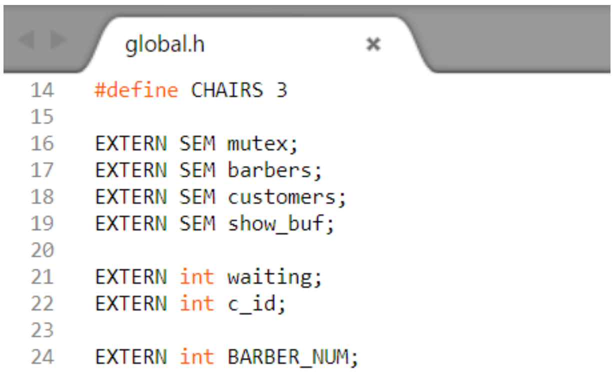

main.c
```c
void add_ID(){
    c_id++;
    show_str("customer NO.");
    disp_int(c_id);
    show_str(" come.\n");
    print_wait();
}
```


main.c
```c
void get_haircut(int id){
    show_str("customer NO.");
    disp_int(id);
    show_str(" ");
    show_str("is getting hair cut\n");
}
```

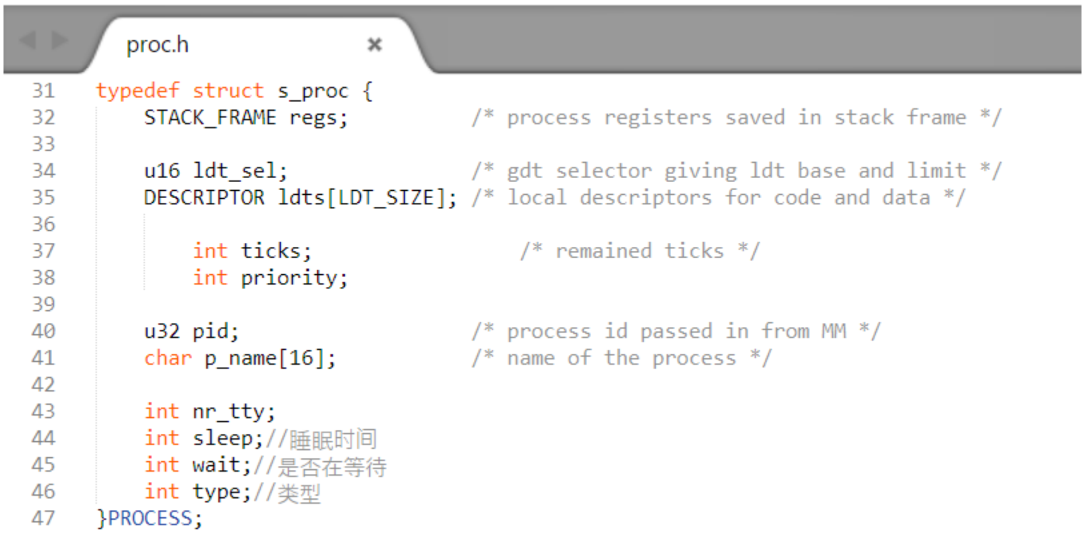
global.c: task_table, user_proc_table 增加进程


global.c
```c
PUBLIC  PROCESS proc_table[NR_TASKS + NR_PROCS];

PUBLIC  TASK    task_table[NR_TASKS] = {
    {task_tty, STACK_SIZE_TTY, "tty"},
    {TestA, STACK_SIZE_TESTA, "TestA"}
};

PUBLIC  TASK    user_proc_table[NR_PROCS] = {
    {TestB, STACK_SIZE_TESTB, "TestB"},
    {TestC, STACK_SIZE_TESTC, "TestC"},
    {TestD, STACK_SIZE_TESTD, "TestD"},
    {TestE, STACK_SIZE_TESTE, "TestE"}
};
```

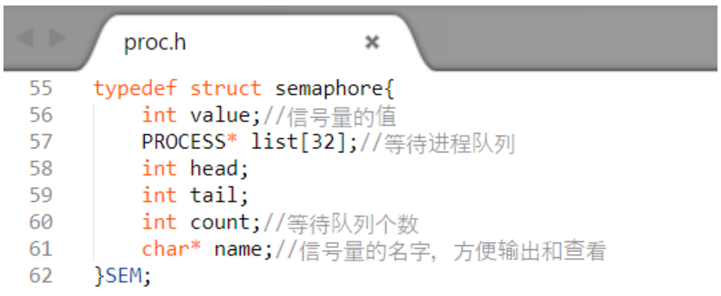
global.h:信号量及一些常量的声明


global.h
```c
#define CHAIRS 3

EXTERN SEM mutex;
EXTERN SEM barbers;
EXTERN SEM customers;
EXTERN SEM show_buf;

EXTERN int waiting;
EXTERN int c_id;

EXTERN int BARBER_NUM;
```

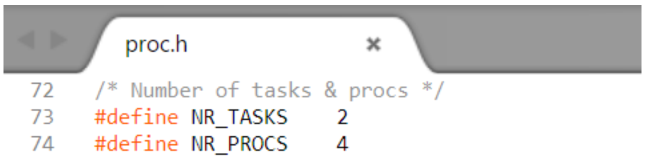
proc.h: PROCESS 中添加属性；SEM 的定义；NR_TASKS 改为 2，NR_PROCS 改为 4；颜色常量的定义；添加任务栈大小。



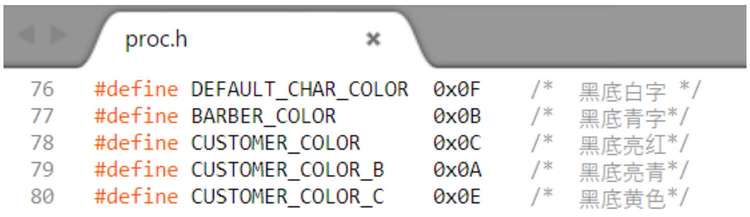
```c
typedef struct s_proc {
    STACK_FRAME regs;          /* process registers saved in stack frame */

    u16 ldt_sel;               /* gdt selector giving ldt base and limit */
    DESCRIPTOR ldts[LDT_SIZE]; /* local descriptors for code and data */

        int ticks;                 /* remained ticks */
        int priority;

    u32 pid;                   /* process id passed in from MM */
    char p_name[16];           /* name of the process */

    int nr_tty;
    int sleep;//睡眠时间
    int wait;//是否在等待
    int type;//类型
}PROCESS;
```



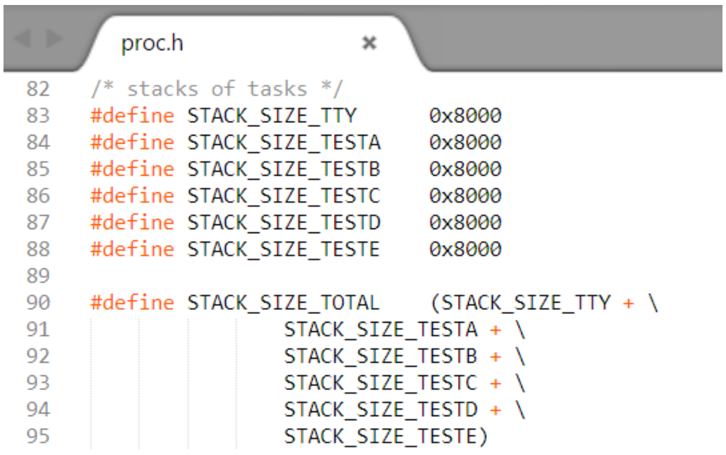
```c
typedef struct semaphore{
    int value;//信号量的值
    PROCESS* list[32];//等待进程队列
    int head;
    int tail;
    int count;//等待队列个数
    char* name;//信号量的名字，方便输出和查看
}SEM;
```



```c
/* Number of tasks & procs */
#define NR_TASKS    2
#define NR_PROCS    4
```

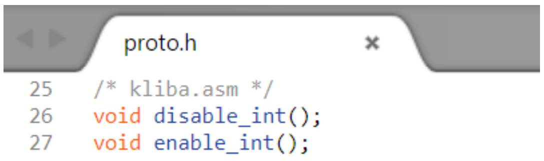


```c
#define DEFAULT_CHAR_COLOR  0x0F    /*  黑底白字 */
#define BARBER_COLOR        0x0B    /*  黑底青字*/
#define CUSTOMER_COLOR      0x0C    /*  黑底亮红*/
#define CUSTOMER_COLOR_B    0x0A    /*  黑底亮青*/
#define CUSTOMER_COLOR_C    0x0E    /*  黑底黄色*/
```

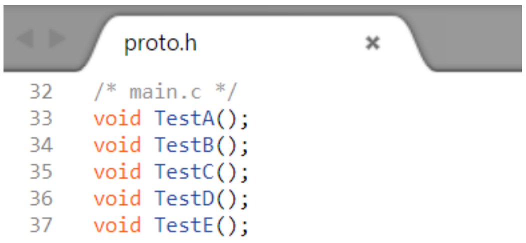


```c
/* stacks of tasks */
#define STACK_SIZE_TTY      0x8000
#define STACK_SIZE_TESTA    0x8000
#define STACK_SIZE_TESTB    0x8000
#define STACK_SIZE_TESTC    0x8000
#define STACK_SIZE_TESTD    0x8000
#define STACK_SIZE_TESTE    0x8000

#define STACK_SIZE_TOTAL    (STACK_SIZE_TTY + \
                STACK_SIZE_TESTA + \
                STACK_SIZE_TESTB + \
                STACK_SIZE_TESTC + \
                STACK_SIZE_TESTD + \
                STACK_SIZE_TESTE)
```

proto.h:进程的声明；开关中断的方法声明

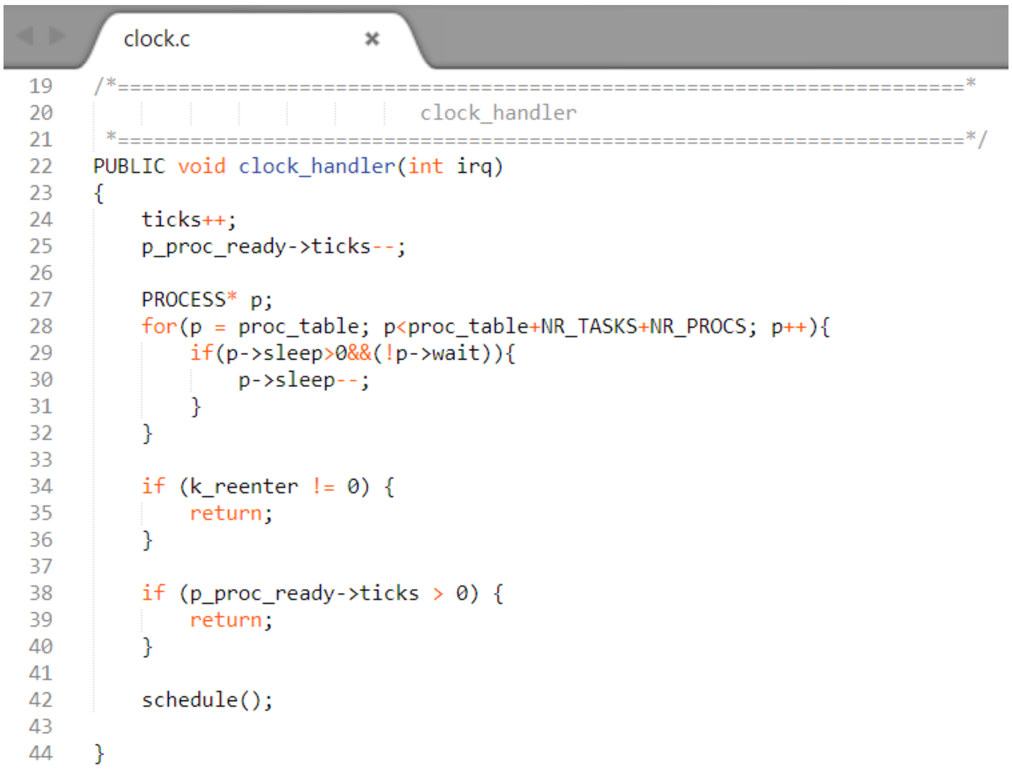


```c
/* kliba.asm */
void disable_int();
void enable_int();
```



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

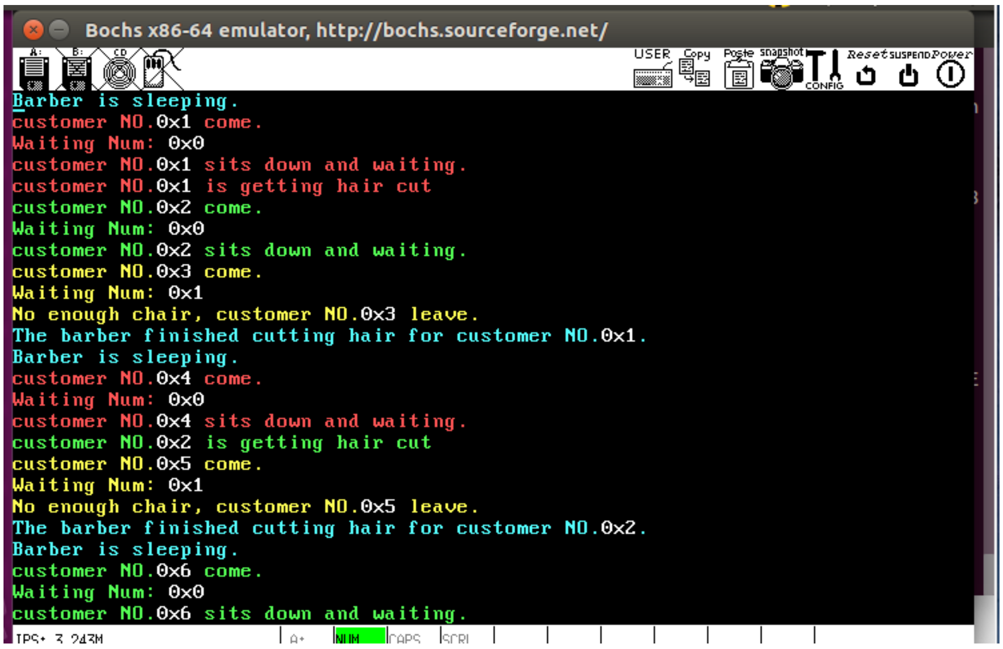
clock.c:时钟的处理



```c
/*======================================================================*
                              clock_handler
 *======================================================================*/
PUBLIC void clock_handler(int irq)
{
    ticks++;
    p_proc_ready->ticks--;

    PROCESS* p;
    for(p = proc_table; p<proc_table+NR_TASKS+NR_PROCS; p++){
        if(p->sleep>0&&(!p->wait)){
            p->sleep--;
        }
    }

    if (k_reenter != 0) {
        return;
    }

    if (p_proc_ready->ticks > 0) {
        return;
    }

    schedule();

}
```

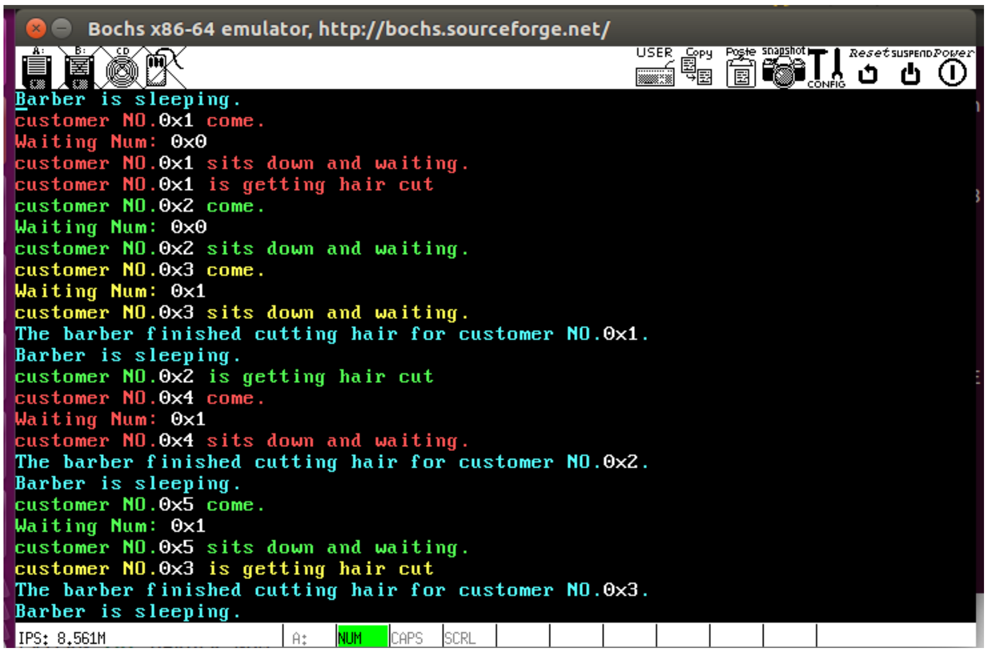
运行截图：

一把椅子



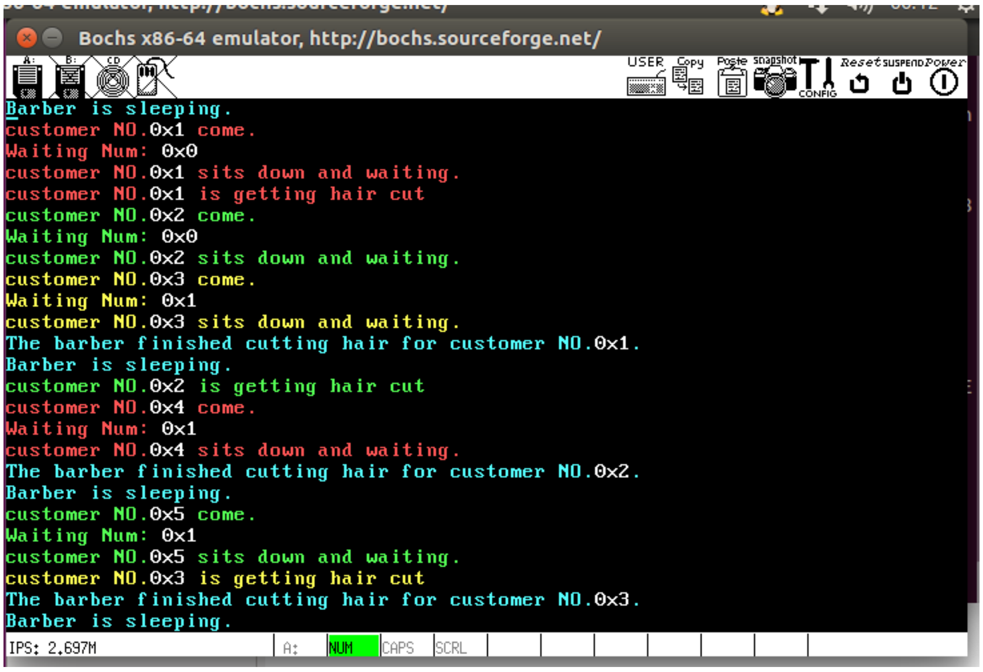
```
Barber is sleeping.
customer NO.0x1 come.
Waiting Num: 0x0
customer NO.0x1 sits down and waiting.
customer NO.0x1 is getting hair cut
customer NO.0x2 come.
Waiting Num: 0x0
customer NO.0x2 sits down and waiting.
customer NO.0x3 come.
Waiting Num: 0x1
No enough chair, customer NO.0x3 leave.
The barber finished cutting hair for customer NO.0x1.
Barber is sleeping.
customer NO.0x4 come.
Waiting Num: 0x0
customer NO.0x4 sits down and waiting.
customer NO.0x2 is getting hair cut
customer NO.0x5 come.
Waiting Num: 0x1
No enough chair, customer NO.0x5 leave.
The barber finished cutting hair for customer NO.0x2.
Barber is sleeping.
customer NO.0x6 come.
Waiting Num: 0x0
customer NO.0x6 sits down and waiting.
```

两把椅子



```
Barber is sleeping.
customer NO.0x1 come.
Waiting Num: 0x0
customer NO.0x1 sits down and waiting.
customer NO.0x1 is getting hair cut
customer NO.0x2 come.
Waiting Num: 0x0
customer NO.0x2 sits down and waiting.
customer NO.0x3 come.
Waiting Num: 0x1
customer NO.0x3 sits down and waiting.
The barber finished cutting hair for customer NO.0x1.
Barber is sleeping.
customer NO.0x2 is getting hair cut
customer NO.0x4 come.
Waiting Num: 0x1
customer NO.0x4 sits down and waiting.
The barber finished cutting hair for customer NO.0x2.
Barber is sleeping.
customer NO.0x5 come.
Waiting Num: 0x1
customer NO.0x5 sits down and waiting.
customer NO.0x3 is getting hair cut
The barber finished cutting hair for customer NO.0x3.
Barber is sleeping.
```

三把椅子



```
Barber is sleeping.
customer NO.0x1 come.
Waiting Num: 0x0
customer NO.0x1 sits down and waiting.
customer NO.0x1 is getting hair cut
customer NO.0x2 come.
Waiting Num: 0x0
customer NO.0x2 sits down and waiting.
customer NO.0x3 come.
Waiting Num: 0x1
customer NO.0x3 sits down and waiting.
The barber finished cutting hair for customer NO.0x1.
Barber is sleeping.
customer NO.0x2 is getting hair cut
customer NO.0x4 come.
Waiting Num: 0x1
customer NO.0x4 sits down and waiting.
The barber finished cutting hair for customer NO.0x2.
Barber is sleeping.
customer NO.0x5 come.
Waiting Num: 0x1
customer NO.0x5 sits down and waiting.
customer NO.0x3 is getting hair cut
The barber finished cutting hair for customer NO.0x3.
Barber is sleeping.
```