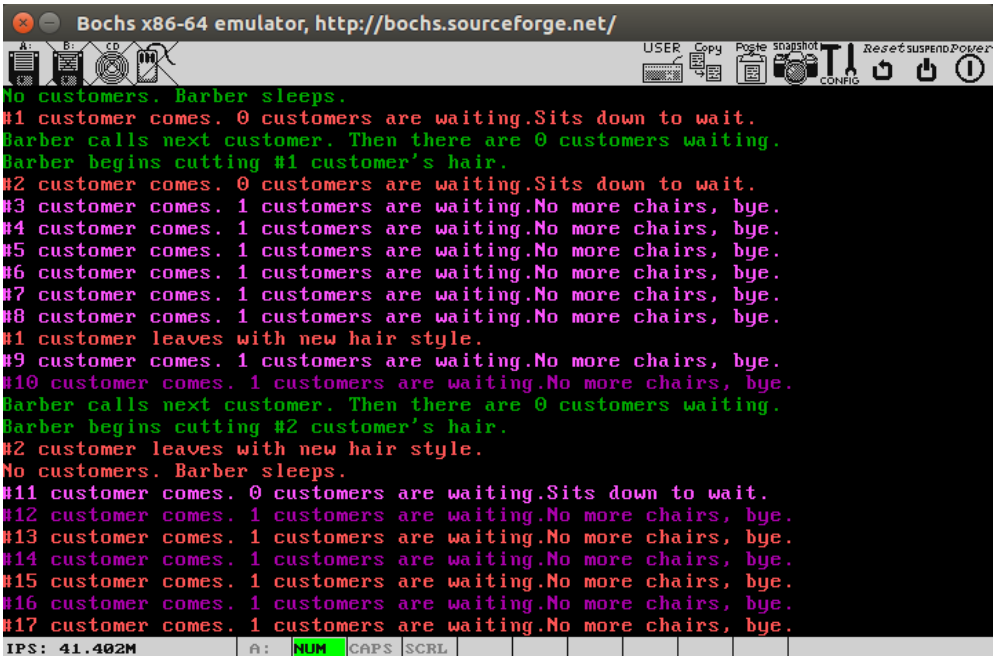
操作系统实验四检查——151250160 吴静琦

运行环境：ubuntu32 位

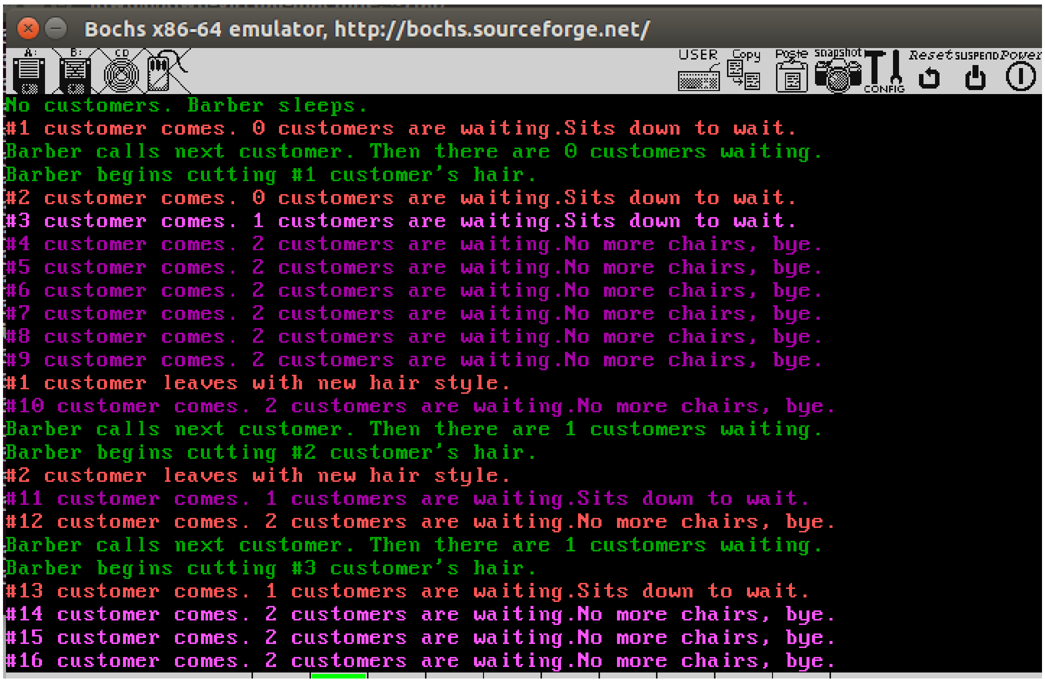
1 运行截图

1.1 一张椅子



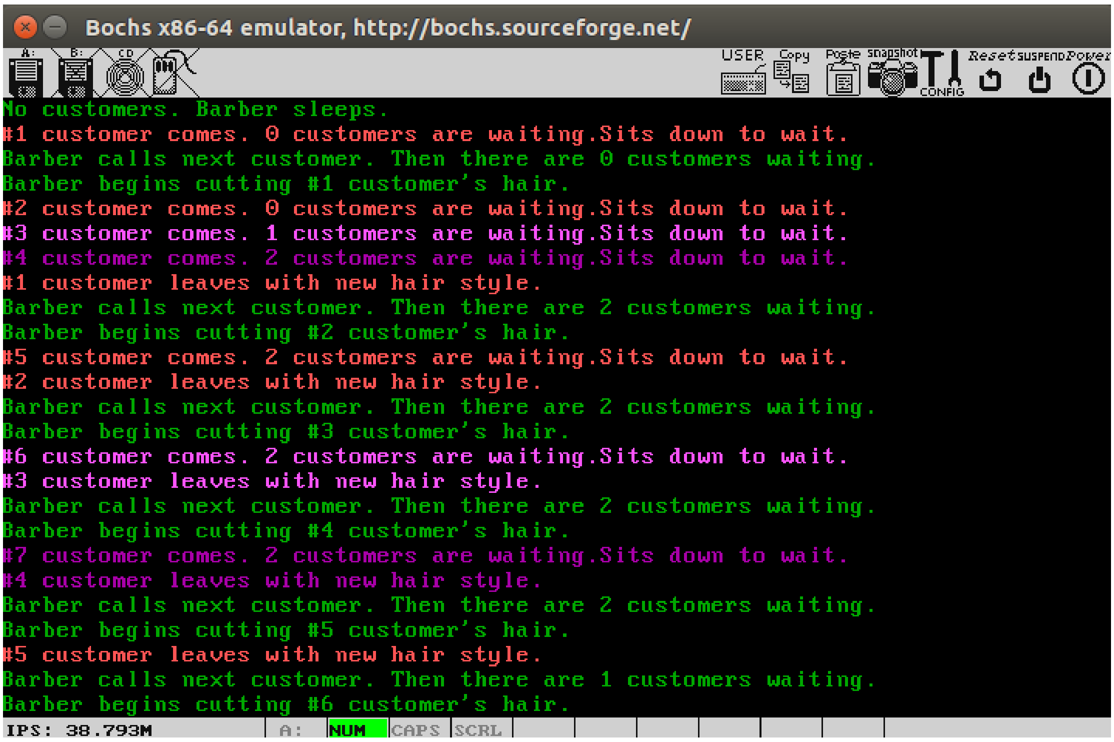
```
No customers. Barber sleeps.
#1 customer comes. 0 customers are waiting.Sits down to wait.
Barber calls next customer. Then there are 0 customers waiting.
Barber begins cutting #1 customer's hair.
#2 customer comes. 0 customers are waiting.Sits down to wait.
#3 customer comes. 1 customers are waiting.No more chairs, bye.
#4 customer comes. 1 customers are waiting.No more chairs, bye.
#5 customer comes. 1 customers are waiting.No more chairs, bye.
#6 customer comes. 1 customers are waiting.No more chairs, bye.
#7 customer comes. 1 customers are waiting.No more chairs, bye.
#8 customer comes. 1 customers are waiting.No more chairs, bye.
#1 customer leaves with new hair style.
#9 customer comes. 1 customers are waiting.No more chairs, bye.
#10 customer comes. 1 customers are waiting.No more chairs, bye.
Barber calls next customer. Then there are 0 customers waiting.
Barber begins cutting #2 customer's hair.
#2 customer leaves with new hair style.
No customers. Barber sleeps.
#11 customer comes. 0 customers are waiting.Sits down to wait.
#12 customer comes. 1 customers are waiting.No more chairs, bye.
#13 customer comes. 1 customers are waiting.No more chairs, bye.
#14 customer comes. 1 customers are waiting.No more chairs, bye.
#15 customer comes. 1 customers are waiting.No more chairs, bye.
#16 customer comes. 1 customers are waiting.No more chairs, bye.
#17 customer comes. 1 customers are waiting.No more chairs, bye.
```

1.2 两张椅子



```
No customers. Barber sleeps.
#1 customer comes. 0 customers are waiting.Sits down to wait.
Barber calls next customer. Then there are 0 customers waiting.
Barber begins cutting #1 customer's hair.
#2 customer comes. 0 customers are waiting.Sits down to wait.
#3 customer comes. 1 customers are waiting.Sits down to wait.
#4 customer comes. 2 customers are waiting.No more chairs, bye.
#5 customer comes. 2 customers are waiting.No more chairs, bye.
#6 customer comes. 2 customers are waiting.No more chairs, bye.
#7 customer comes. 2 customers are waiting.No more chairs, bye.
#8 customer comes. 2 customers are waiting.No more chairs, bye.
#9 customer comes. 2 customers are waiting.No more chairs, bye.
#1 customer leaves with new hair style.
#10 customer comes. 2 customers are waiting.No more chairs, bye.
Barber calls next customer. Then there are 1 customers waiting.
Barber begins cutting #2 customer's hair.
#2 customer leaves with new hair style.
#11 customer comes. 1 customers are waiting.Sits down to wait.
#12 customer comes. 2 customers are waiting.No more chairs, bye.
Barber calls next customer. Then there are 1 customers waiting.
Barber begins cutting #3 customer's hair.
#13 customer comes. 1 customers are waiting.Sits down to wait.
#14 customer comes. 2 customers are waiting.No more chairs, bye.
#15 customer comes. 2 customers are waiting.No more chairs, bye.
#16 customer comes. 2 customers are waiting.No more chairs, bye.
```

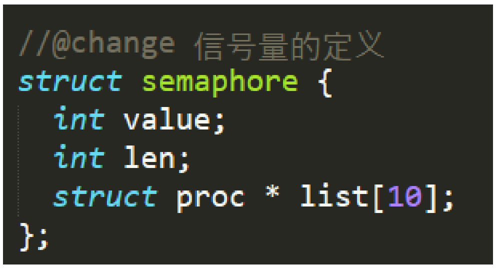
1.3 三张椅子



```
No customers. Barber sleeps.
#1 customer comes. 0 customers are waiting.Sits down to wait.
Barber calls next customer. Then there are 0 customers waiting.
Barber begins cutting #1 customer's hair.
#2 customer comes. 0 customers are waiting.Sits down to wait.
#3 customer comes. 1 customers are waiting.Sits down to wait.
#4 customer comes. 2 customers are waiting.Sits down to wait.
#1 customer leaves with new hair style.
Barber calls next customer. Then there are 2 customers waiting.
Barber begins cutting #2 customer's hair.
#5 customer comes. 2 customers are waiting.Sits down to wait.
#2 customer leaves with new hair style.
Barber calls next customer. Then there are 2 customers waiting.
Barber begins cutting #3 customer's hair.
#6 customer comes. 2 customers are waiting.Sits down to wait.
#3 customer leaves with new hair style.
Barber calls next customer. Then there are 2 customers waiting.
Barber begins cutting #4 customer's hair.
#7 customer comes. 2 customers are waiting.Sits down to wait.
#4 customer leaves with new hair style.
Barber calls next customer. Then there are 2 customers waiting.
Barber begins cutting #5 customer's hair.
#5 customer leaves with new hair style.
Barber calls next customer. Then there are 1 customers waiting.
Barber begins cutting #6 customer's hair.
```

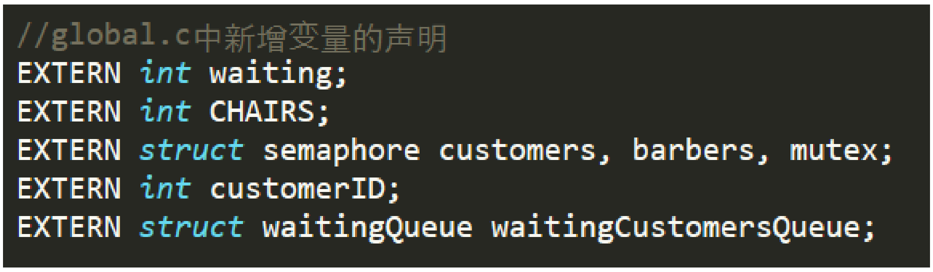
2　新增的变量定义

2.1 修改 proc.h



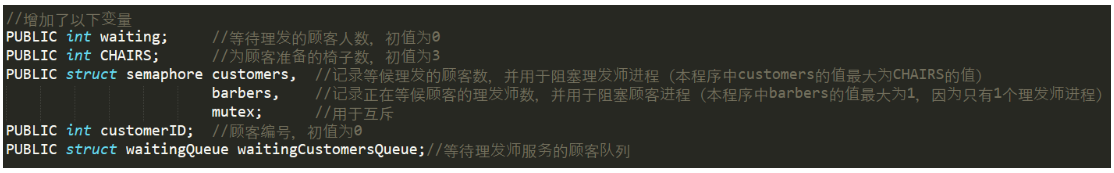
```c
//@change 信号量的定义
struct semaphore {
  int value;
  int len;
  struct proc * list[10];
};
```

2.2 修改 global.h



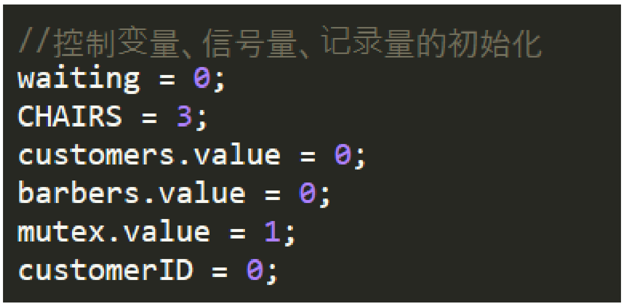
```c
//global.c中新增变量的声明
EXTERN int waiting;
EXTERN int CHAIRS;
EXTERN struct semaphore customers, barbers, mutex;
EXTERN int customerID;
EXTERN struct waitingQueue waitingCustomersQueue;
```

2.3 修改 global.c



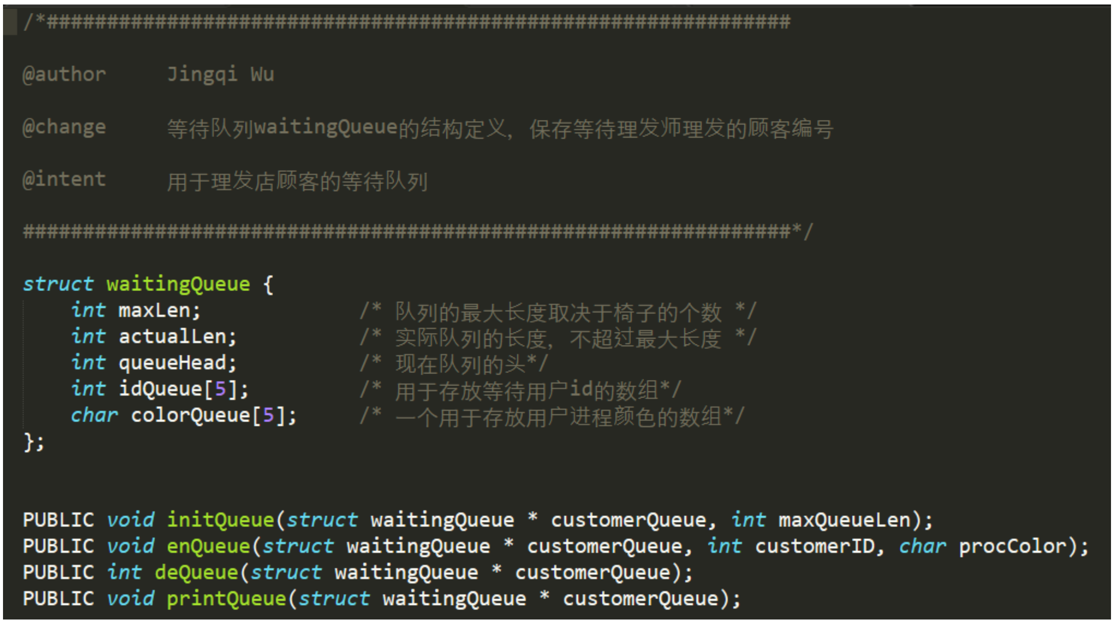
```c
//增加了以下变量
PUBLIC int waiting;      //等待理发的顾客人数，初值为0
PUBLIC int CHAIRS;       //为顾客准备的椅子数，初值为3
PUBLIC struct semaphore customers,  //记录等候理发的顾客数，并用于阻塞理发师进程（本程序中customers的值最大为CHAIRS的值）
                        barbers,    //记录正在等候顾客的理发师数，并用于阻塞顾客进程（本程序中barbers的值最大为1，因为只有1个理发师进程）
                        mutex;      //用于互斥
PUBLIC int customerID;  //顾客编号，初值为0
PUBLIC struct waitingQueue waitingCustomersQueue;//等待理发师服务的顾客队列
```

2.4 修改 main.c

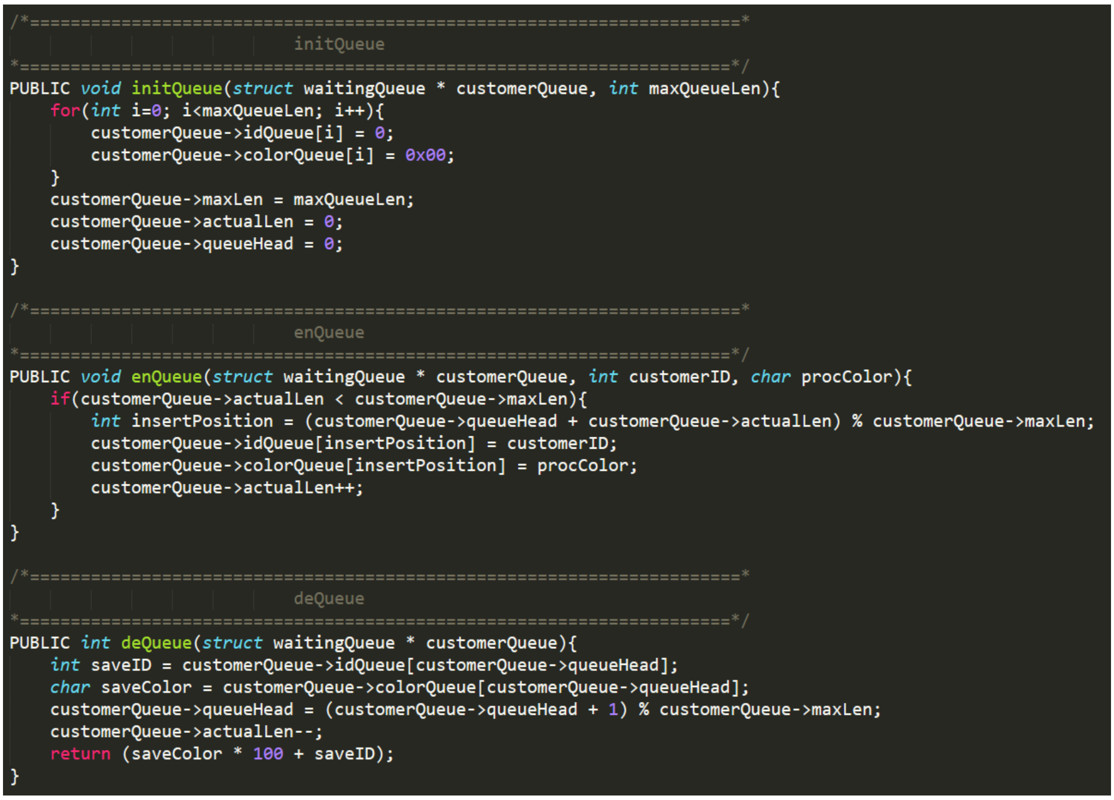


```c
//控制变量、信号量、记录量的初始化
waiting = 0;
CHAIRS = 3;
customers.value = 0;
barbers.value = 0;
mutex.value = 1;
customerID = 0;
```

2.5 新增 queue.h，queue.c



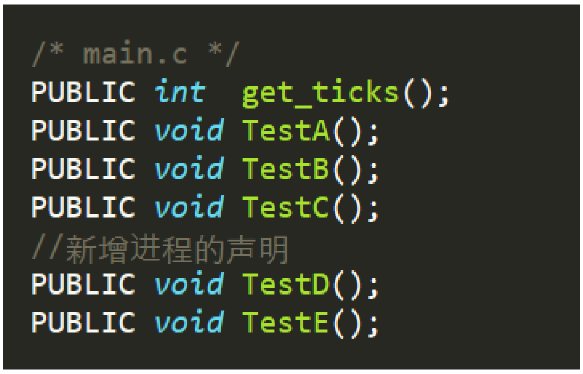
```c
/*#################################################################

@author     Jingqi Wu

@change     等待队列waitingQueue的结构定义，保存等待理发师理发的顾客编号

@intent     用于理发店顾客的等待队列

#################################################################*/

struct waitingQueue {
    int maxLen;             /* 队列的最大长度取决于椅子的个数 */
    int actualLen;          /* 实际队列的长度，不超过最大长度 */
    int queueHead;          /* 现在队列的头*/
    int idQueue[5];         /* 用于存放等待用户id的数组*/
    char colorQueue[5];     /* 一个用于存放用户进程颜色的数组*/
};


PUBLIC void initQueue(struct waitingQueue * customerQueue, int maxQueueLen);
PUBLIC void enQueue(struct waitingQueue * customerQueue, int customerID, char procColor);
PUBLIC int deQueue(struct waitingQueue * customerQueue);
PUBLIC void printQueue(struct waitingQueue * customerQueue);
```



```c
/*======================================================================*
                              initQueue
*======================================================================*/
PUBLIC void initQueue(struct waitingQueue * customerQueue, int maxQueueLen){
    for(int i=0; i<maxQueueLen; i++){
        customerQueue->idQueue[i] = 0;
        customerQueue->colorQueue[i] = 0x00;
    }
    customerQueue->maxLen = maxQueueLen;
    customerQueue->actualLen = 0;
    customerQueue->queueHead = 0;
}

/*======================================================================*
                              enQueue
*======================================================================*/
PUBLIC void enQueue(struct waitingQueue * customerQueue, int customerID, char procColor){
    if(customerQueue->actualLen < customerQueue->maxLen){
        int insertPosition = (customerQueue->queueHead + customerQueue->actualLen) % customerQueue->maxLen;
        customerQueue->idQueue[insertPosition] = customerID;
        customerQueue->colorQueue[insertPosition] = procColor;
        customerQueue->actualLen++;
    }
}

/*======================================================================*
                              deQueue
*======================================================================*/
PUBLIC int deQueue(struct waitingQueue * customerQueue){
    int saveID = customerQueue->idQueue[customerQueue->queueHead];
    char saveColor = customerQueue->colorQueue[customerQueue->queueHead];
    customerQueue->queueHead = (customerQueue->queueHead + 1) % customerQueue->maxLen;
    customerQueue->actualLen--;
    return (saveColor * 100 + saveID);
}
```

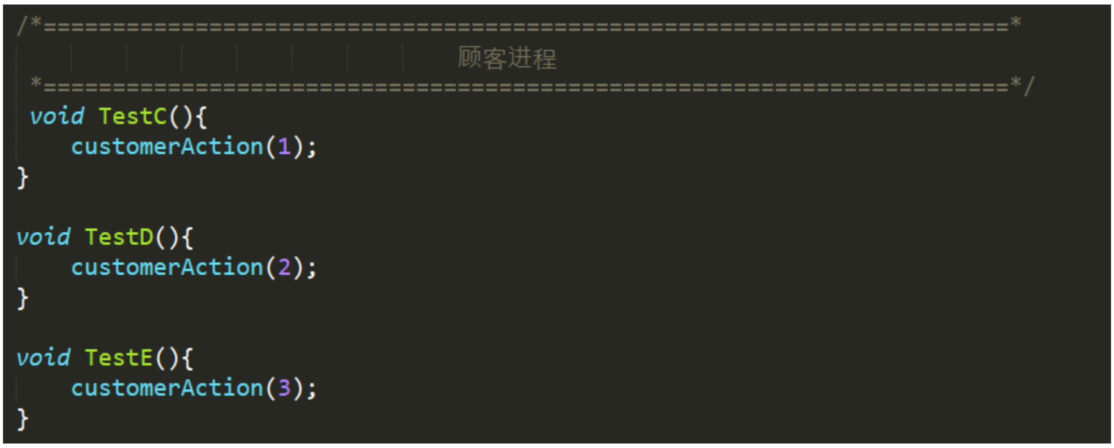
3　新增 D、E 两个进程

3.1 修改 proto.h



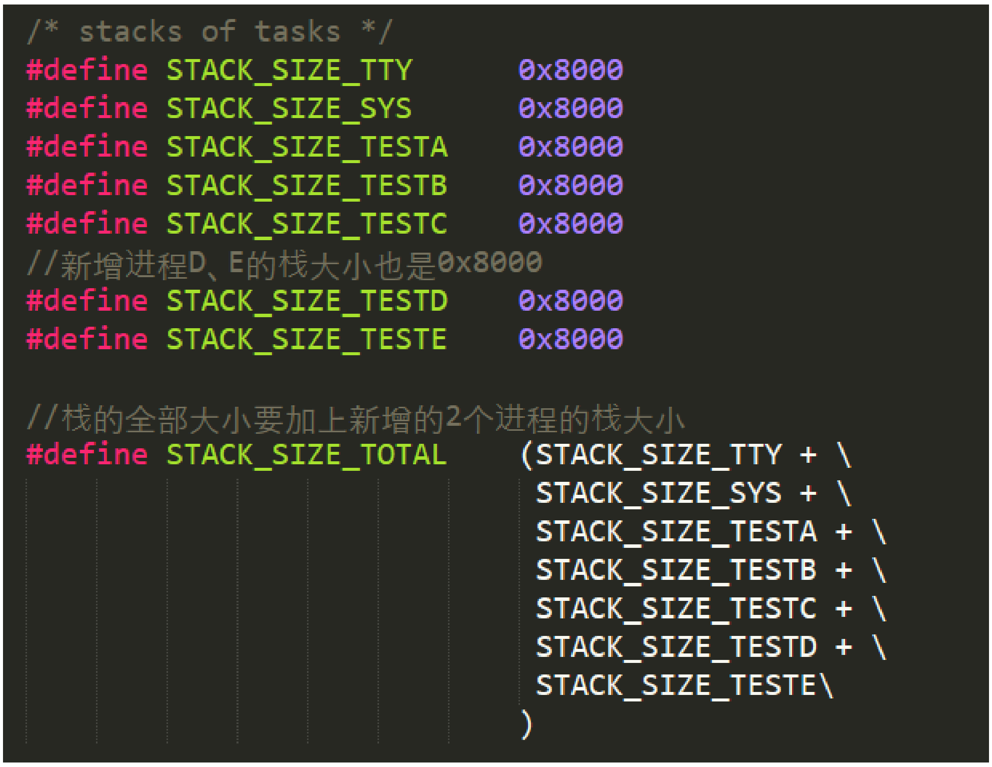
```
/* main.c */
PUBLIC int  get_ticks();
PUBLIC void TestA();
PUBLIC void TestB();
PUBLIC void TestC();
//新增进程的声明
PUBLIC void TestD();
PUBLIC void TestE();
```

3.2 修改 main.c



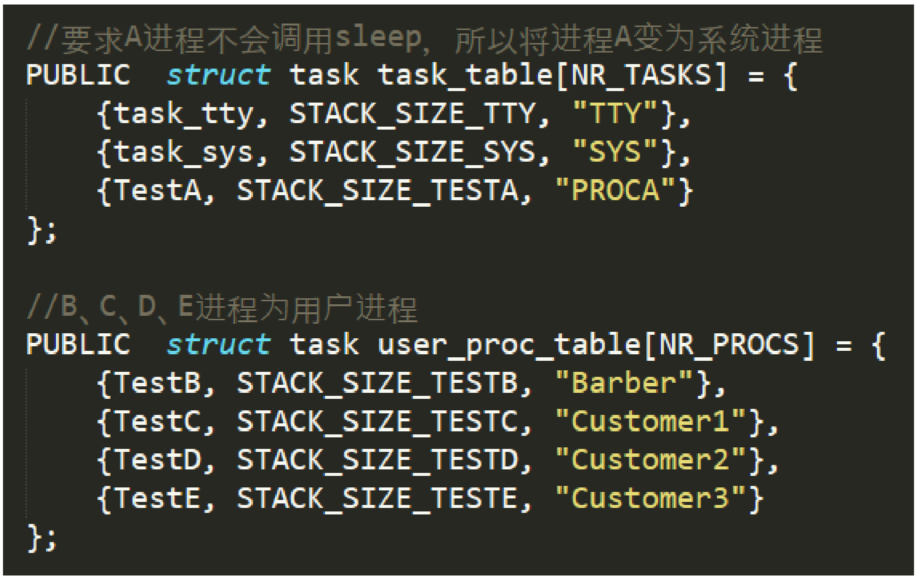
```
/*======================================================================*
                                顾客进程
 *======================================================================*/
 void TestC(){
    customerAction(1);
}

void TestD(){
    customerAction(2);
}

void TestE(){
    customerAction(3);
}
```

3.3 修改 proc.h



```
/* stacks of tasks */
#define STACK_SIZE_TTY      0x8000
#define STACK_SIZE_SYS      0x8000
#define STACK_SIZE_TESTA    0x8000
#define STACK_SIZE_TESTB    0x8000
#define STACK_SIZE_TESTC    0x8000
//新增进程D、E的栈大小也是0x8000
#define STACK_SIZE_TESTD    0x8000
#define STACK_SIZE_TESTE    0x8000

//栈的全部大小要加上新增的2个进程的栈大小
#define STACK_SIZE_TOTAL    (STACK_SIZE_TTY + \
                             STACK_SIZE_SYS + \
                             STACK_SIZE_TESTA + \
                             STACK_SIZE_TESTB + \
                             STACK_SIZE_TESTC + \
                             STACK_SIZE_TESTD + \
                             STACK_SIZE_TESTE\
                            )
```

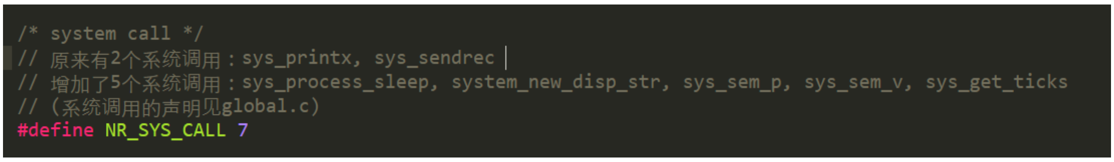
3.4 修改 global.c



```
//要求A进程不会调用sleep，所以将进程A变为系统进程
PUBLIC  struct task task_table[NR_TASKS] = {
    {task_tty, STACK_SIZE_TTY, "TTY"},
    {task_sys, STACK_SIZE_SYS, "SYS"},
    {TestA, STACK_SIZE_TESTA, "PROCA"}
};

//B、C、D、E进程为用户进程
PUBLIC  struct task user_proc_table[NR_PROCS] = {
    {TestB, STACK_SIZE_TESTB, "Barber"},
    {TestC, STACK_SIZE_TESTC, "Customer1"},
    {TestD, STACK_SIZE_TESTD, "Customer2"},
    {TestE, STACK_SIZE_TESTE, "Customer3"}
};
```

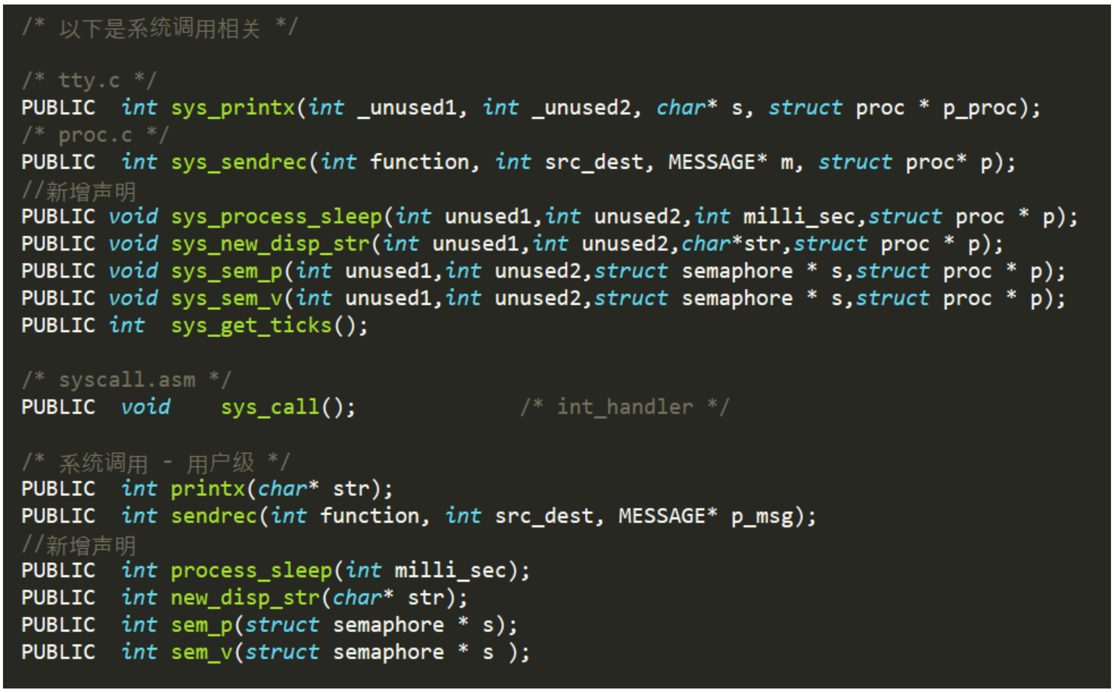
4　新增四个系统调用

4.1 修改 const.h



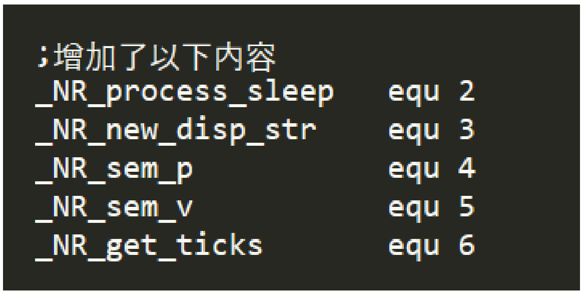
```
/* system call */
// 原来有2个系统调用：sys_printx, sys_sendrec
// 增加了5个系统调用：sys_process_sleep, system_new_disp_str, sys_sem_p, sys_sem_v, sys_get_ticks
// (系统调用的声明见global.c)
#define NR_SYS_CALL 7
```

4.2 修改 proto.h

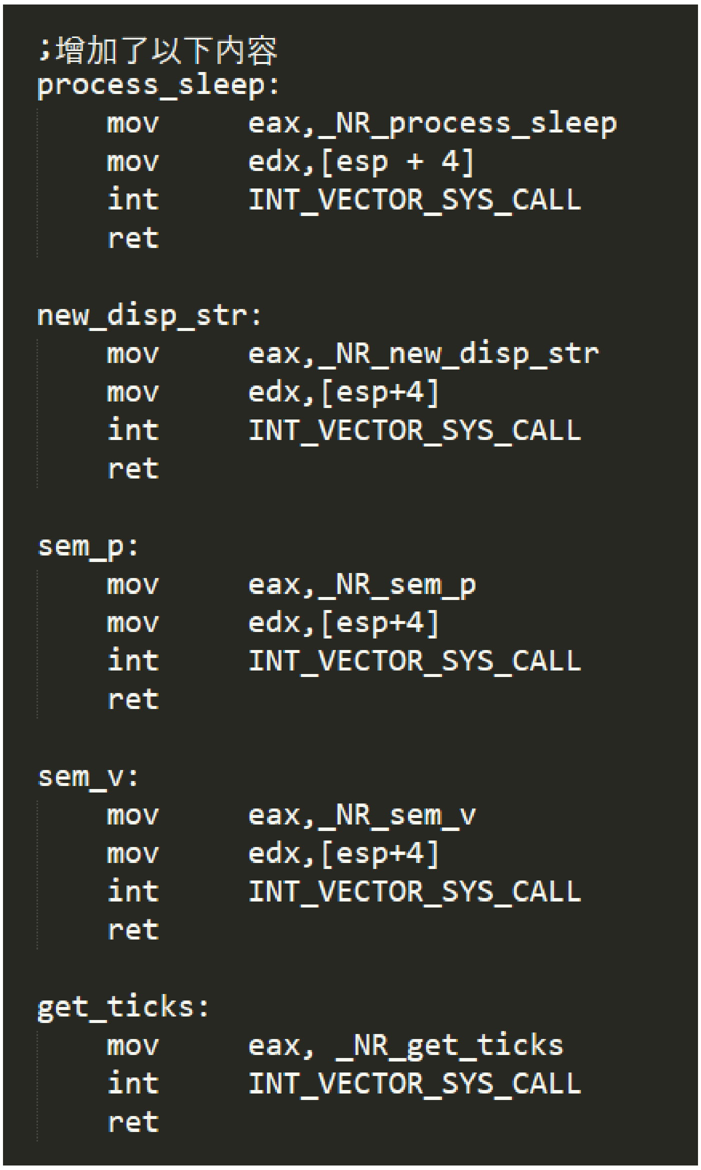


```
/* 以下是系统调用相关 */

/* tty.c */
PUBLIC  int sys_printx(int _unused1, int _unused2, char* s, struct proc * p_proc);
/* proc.c */
PUBLIC  int sys_sendrec(int function, int src_dest, MESSAGE* m, struct proc* p);
//新增声明
PUBLIC void sys_process_sleep(int unused1,int unused2,int milli_sec,struct proc * p);
PUBLIC void sys_new_disp_str(int unused1,int unused2,char*str,struct proc * p);
PUBLIC void sys_sem_p(int unused1,int unused2,struct semaphore * s,struct proc * p);
PUBLIC void sys_sem_v(int unused1,int unused2,struct semaphore * s,struct proc * p);
PUBLIC int  sys_get_ticks();

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

/* 系统调用 - 用户级 */
PUBLIC  int printx(char* str);
PUBLIC  int sendrec(int function, int src_dest, MESSAGE* p_msg);
//新增声明
PUBLIC  int process_sleep(int milli_sec);
PUBLIC  int new_disp_str(char* str);
PUBLIC  int sem_p(struct semaphore * s);
PUBLIC  int sem_v(struct semaphore * s );
```

4.3 修改 syscall.asm

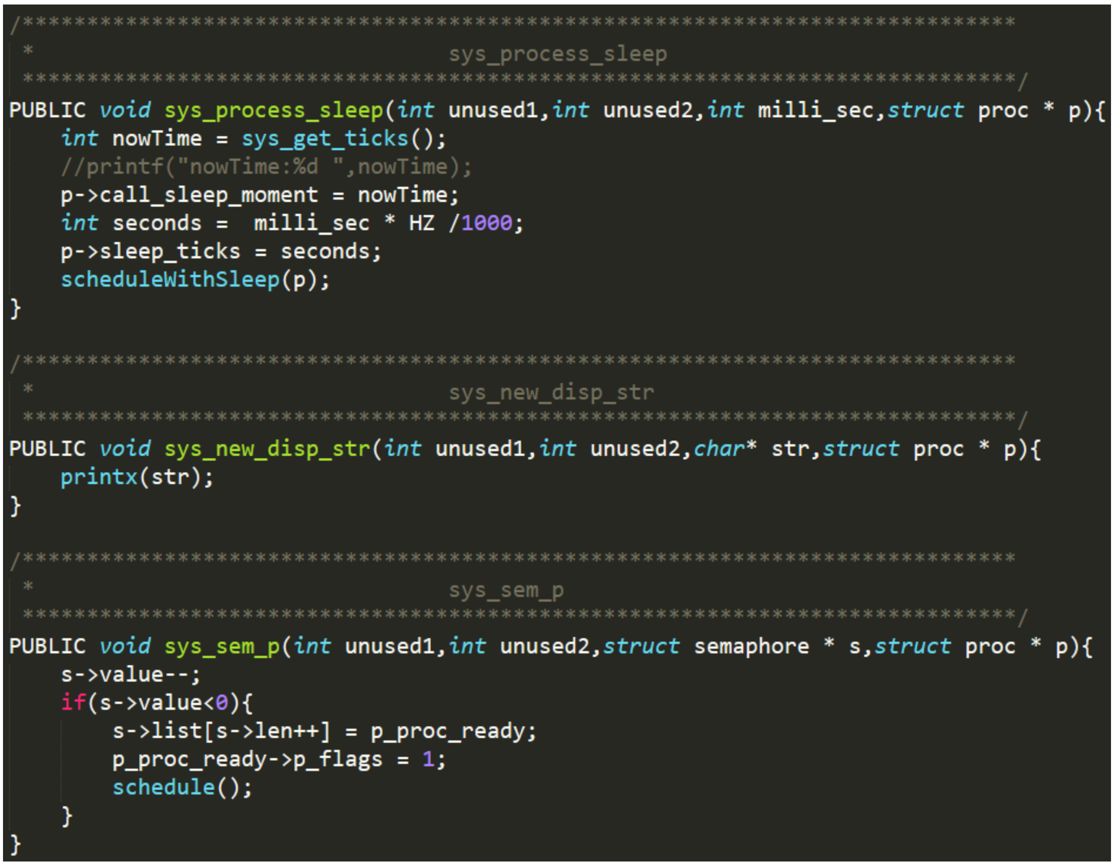


```
;增加了以下内容
_NR_process_sleep   equ 2
_NR_new_disp_str    equ 3
_NR_sem_p           equ 4
_NR_sem_v           equ 5
_NR_get_ticks       equ 6
```

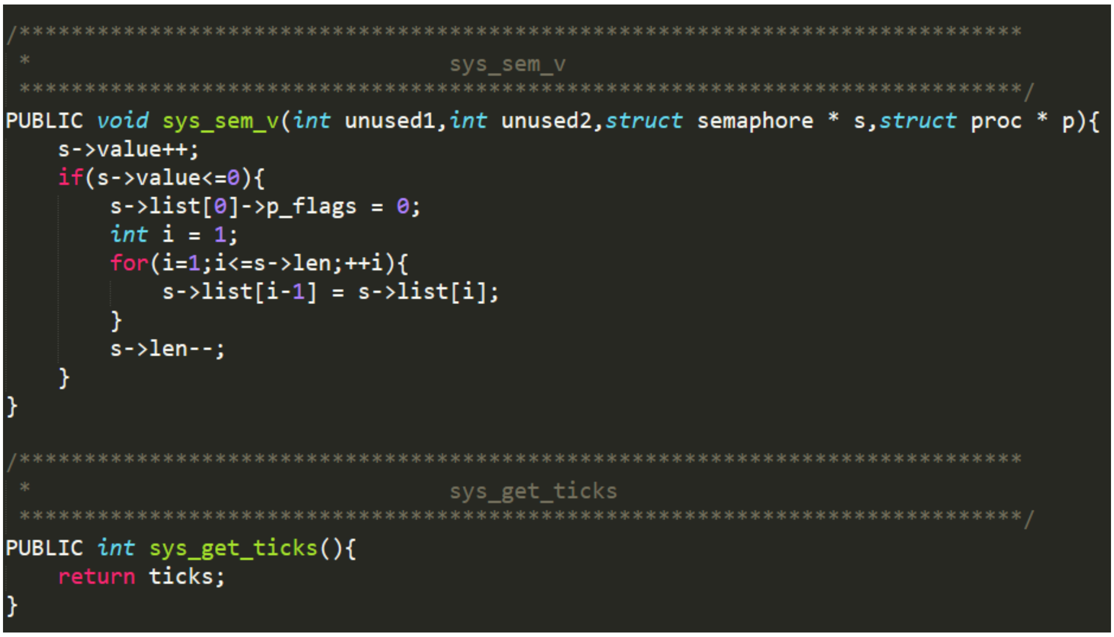


```
;增加了以下内容
process_sleep:
    mov     eax,_NR_process_sleep
    mov     edx,[esp + 4]
    int     INT_VECTOR_SYS_CALL
    ret

new_disp_str:
    mov     eax,_NR_new_disp_str
    mov     edx,[esp+4]
    int     INT_VECTOR_SYS_CALL
    ret

sem_p:
    mov     eax,_NR_sem_p
    mov     edx,[esp+4]
    int     INT_VECTOR_SYS_CALL
    ret

sem_v:
    mov     eax,_NR_sem_v
    mov     edx,[esp+4]
    int     INT_VECTOR_SYS_CALL
    ret

get_ticks:
    mov     eax, _NR_get_ticks
    int     INT_VECTOR_SYS_CALL
    ret
```

4.4 修改 proc.c

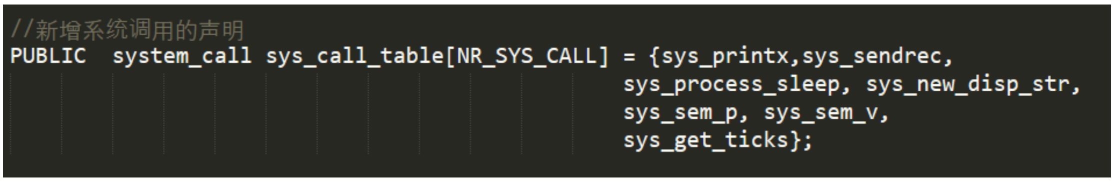


```c
/*****************************************************************************
 *                                sys_process_sleep
 *****************************************************************************/
PUBLIC void sys_process_sleep(int unused1,int unused2,int milli_sec,struct proc * p){
    int nowTime = sys_get_ticks();
    //printf("nowTime:%d ",nowTime);
    p->call_sleep_moment = nowTime;
    int seconds =  milli_sec * HZ /1000;
    p->sleep_ticks = seconds;
    scheduleWithSleep(p);
}

/*****************************************************************************
 *                                sys_new_disp_str
 *****************************************************************************/
PUBLIC void sys_new_disp_str(int unused1,int unused2,char* str,struct proc * p){
    printx(str);
}

/*****************************************************************************
 *                                sys_sem_p
 *****************************************************************************/
PUBLIC void sys_sem_p(int unused1,int unused2,struct semaphore * s,struct proc * p){
    s->value--;
    if(s->value<0){
        s->list[s->len++] = p_proc_ready;
        p_proc_ready->p_flags = 1;
        schedule();
    }
}
```



```c
/*****************************************************************************
 *                                sys_sem_v
 *****************************************************************************/
PUBLIC void sys_sem_v(int unused1,int unused2,struct semaphore * s,struct proc * p){
    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--;
    }
}

/*****************************************************************************
 *                                sys_get_ticks
 *****************************************************************************/
PUBLIC int sys_get_ticks(){
    return ticks;
}
```

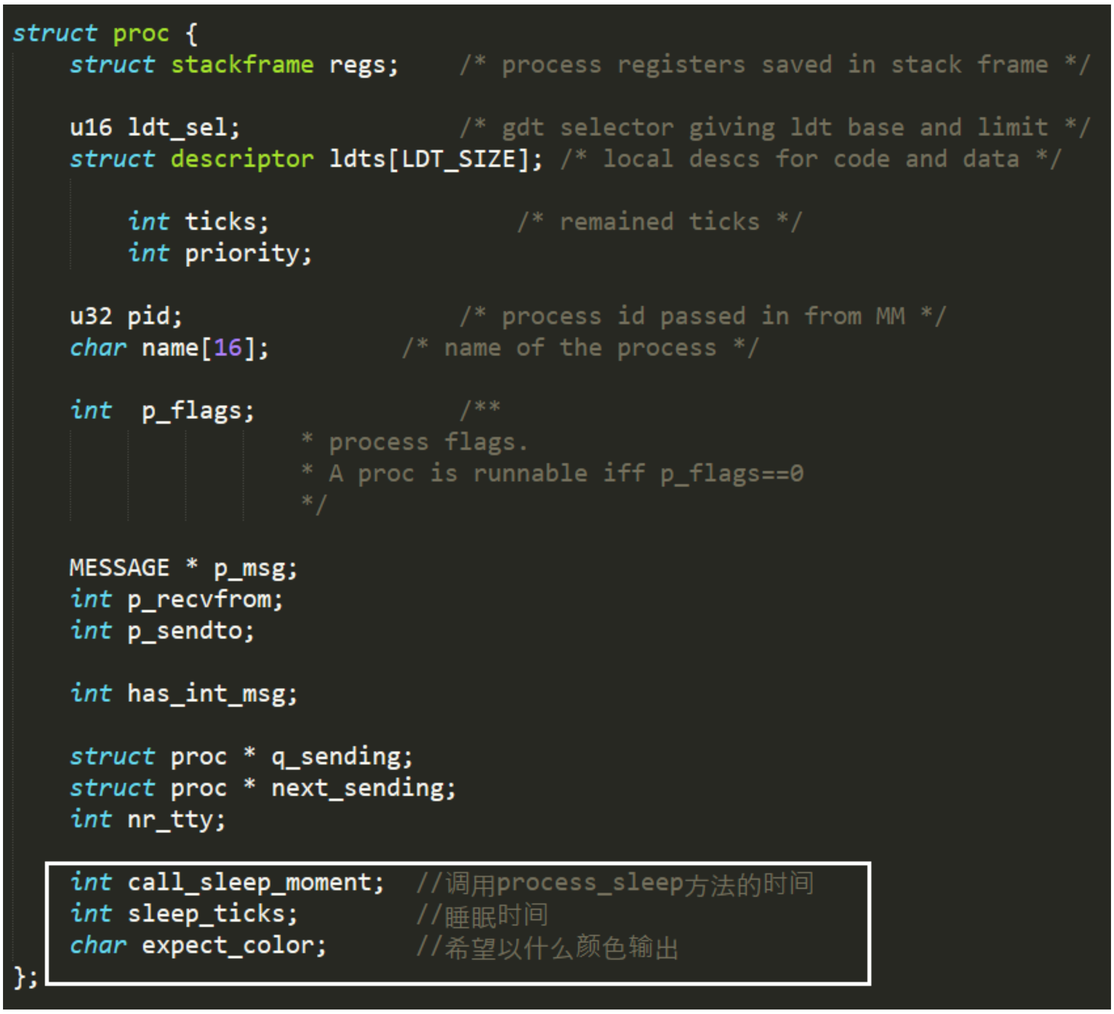
4.5 修改 global.c



```
//新增系统调用的声明
PUBLIC  system_call sys_call_table[NR_SYS_CALL] = {sys_printx,sys_sendrec,
                                                sys_process_sleep, sys_new_disp_str,
                                                sys_sem_p, sys_sem_v,
                                                sys_get_ticks};
```

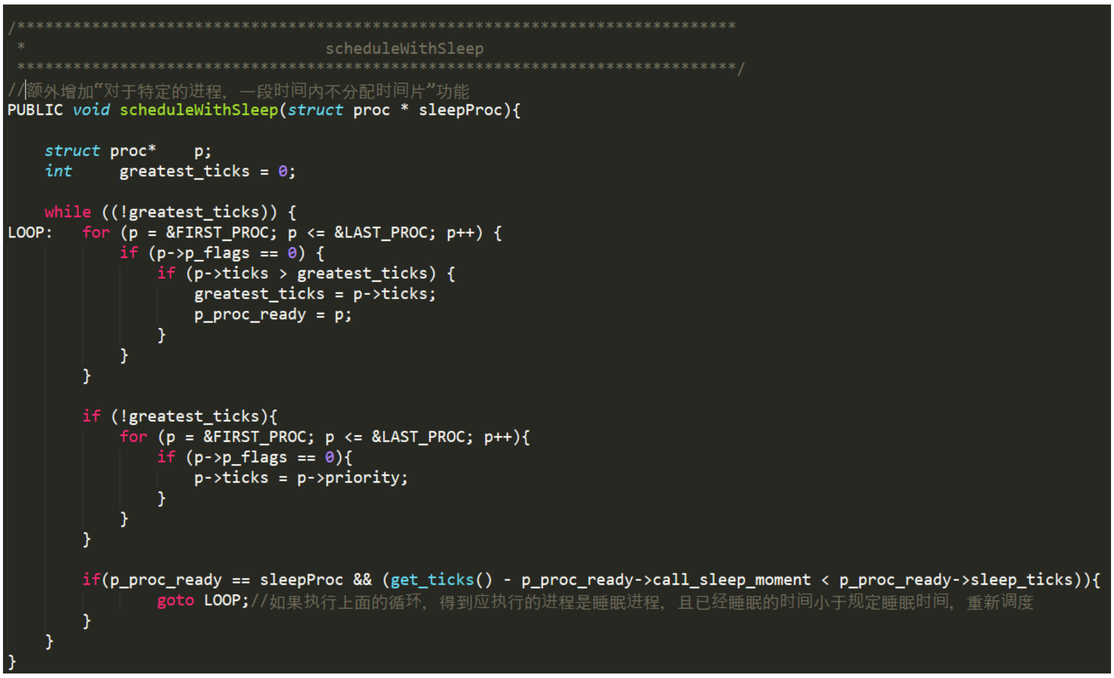
5 进程睡眠的实现

5.1 修改 proc.h



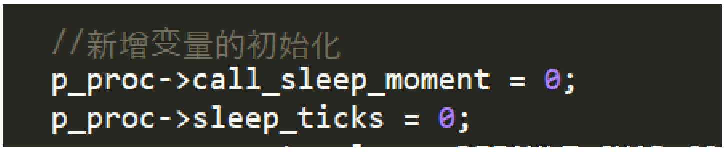
```
struct proc {
    struct stackframe regs;    /* process registers saved in stack frame */

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

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

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

    int  p_flags;              /**
                    * process flags.
                    * A proc is runnable iff p_flags==0
                    */

    MESSAGE * p_msg;
    int p_recvfrom;
    int p_sendto;

    int has_int_msg;

    struct proc * q_sending;
    struct proc * next_sending;
    int nr_tty;

    int call_sleep_moment;  //调用process_sleep方法的时间
    int sleep_ticks;        //睡眠时间
    char expect_color;      //希望以什么颜色输出
};
```

5.2 修改 proc.c



```
/*****************************************************************************
 *                               scheduleWithSleep
 *****************************************************************************/
//额外增加“对于特定的进程，一段时间内不分配时间片”功能
PUBLIC void scheduleWithSleep(struct proc * sleepProc){

    struct proc*    p;
    int     greatest_ticks = 0;

    while ((!greatest_ticks)) {
LOOP:   for (p = &FIRST_PROC; p <= &LAST_PROC; p++) {
            if (p->p_flags == 0) {
                if (p->ticks > greatest_ticks) {
                    greatest_ticks = p->ticks;
                    p_proc_ready = p;
                }
            }
        }

        if (!greatest_ticks){
            for (p = &FIRST_PROC; p <= &LAST_PROC; p++){
                if (p->p_flags == 0){
                    p->ticks = p->priority;
                }
            }
        }

        if(p_proc_ready == sleepProc && (get_ticks() - p_proc_ready->call_sleep_moment < p_proc_ready->sleep_ticks)){
                goto LOOP;//如果执行上面的循环，得到应执行的进程是睡眠进程，且已经睡眠的时间小于规定睡眠时间，重新调度
        }
    }
}
```

5.3 修改 main.c



```
//新增变量的初始化
p_proc->call_sleep_moment = 0;
p_proc->sleep_ticks = 0;
```