运行环境 ubuntu-mate-16.10-32 32 位

一、增加系统调用

1、修改 syscall.asm

增加系统调用函数声明和定义：

```
;添加系统调用的对外接口
INT_VECTOR_SYS_CALL equ 0x90
_NR_get_ticks                   equ 0 ; 要跟 global.c 中 sys_call_table
_NR_sleep                       equ 1 ;
_NR_my_disp                     equ 2 ;
_NR_P_operate                   equ 3 ;
_NR_V_operate                   equ 4 ;
_NR_write                   equ 5 ;

; 导出符号
global  write
global  get_ticks
global  sleep
global  P_operate
global  V_operate
global  my_disp

bits 32
[section .text]

; ====================================================================
;                                get_ticks
; ====================================================================
get_ticks:
    mov eax, _NR_get_ticks
    int INT_VECTOR_SYS_CALL
    ret

sleep:
    mov ebx,[esp + 4]
    mov eax, _NR_sleep
    int INT_VECTOR_SYS_CALL
    ret

my_disp:
    mov ebx,[esp + 4]
    mov ecx,[esp + 8]
    mov edi,[esp + 12]
```

2、修改 kernel.asm

在 sys_call 中把参数压栈：

```
; ========================================================================
;                                  sys_call
; ========================================================================
sys_call:
        call    save
    push    dword [p_proc_ready]
        sti

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

        cli
        ret
```

3、修改 proto.h

增加对四个函数的声明：

```
/* 系统调用 - 系统级 */
/* proc.c */
PUBLIC  int     sys_get_ticks();
PUBLIC  int     sys_write(char* buf, int len, PROCESS* p_proc);
PUBLIC  int     sys_process_sleep(int milli_delay);
PUBLIC  int     sys_sem_v(SEMAPHORE* s);
PUBLIC  int     sys_sem_p(SEMAPHORE* s);
PUBLIC  int     sys_disp_str(char* str, int length , int color);
/* syscall.asm */
PUBLIC  void    sys_call();             /* int_handler */

/* 系统调用 - 用户级 */
PUBLIC  int     get_ticks();
PUBLIC  void    write(char* buf, int len);
PUBLIC  int     sleep(int milli_delay);
PUBLIC  int     my_disp(char* str, int length , int color);
PUBLIC  int     P_operate(SEMAPHORE* s);
PUBLIC  int     V_operate(SEMAPHORE* s);
```

4、修改 const.h

把系统调用的函数加四：

```
#define AT_WINI_IRQ 14  /* at winche

/* system call */
#define NR_SYS_CALL     6

#endif /* _ORANGES_CONST_H_ */
```

5、修改 global.c：

在 sys_call_table[]中对应添加系统调用：

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

6、在 **proc.c** 中添加对四个系统调用的实现。

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

PUBLIC int sys_process_sleep(int milli_delay)
{

PUBLIC int sys_sem_p(SEMAPHORE* s)
{

PUBLIC int sys_sem_v(SEMAPHORE* s)
{

PUBLIC int sys_disp_str(char* str, int length , int color)
{
```

二、添加用户进程

1、在 main.c 中增加所用的进程

```
void TestC()//costomer C
{
    char output[4];
    char wait[4];
    while(1){
        sleep(3000);

        int numberB = 0;
        P_operate(&number);
        numberB = numbers;
        itoa(output, numberB);
        numbers ++;
        V_operate(&number);

        P_operate(&mutex);
        if (waiting < chairs) {
            waiting++;
            itoa(wait, waiting);
            my_disp("C:customer " , 11 , 0x0D);
            my_disp(output, 4, 0x0D);
            my_disp(" come , add waiting num to: ", 28 , 0x0D);
            my_disp(wait, 4, 0x0D);
            my_disp("\n", 1, 0x0F);
            milli_delay(5000);
            V_operate(&customers);
            V_operate(&mutex);
            P_operate(&barber);
            my_disp("C:customer ", 11, 0x0D);
            my_disp(output, 4, 0x0D);
            my_disp(" get hair cut , leave! \n", 24, 0x0D);
            milli_delay(2000);
        }else{
            my_disp("C:customer ", 11, 0x0D);
            my_disp(output, 4, 0x0D);
            my_disp(" come, leave without hair cut!\n", 31, 0x0D);
            milli_delay(2000);
            V_operate(&mutex);
        }
```

2、在 global.c 的 task_table 中新增所用进程

```
PUBLIC  TASK    user_proc_table[NR_PROCS] = {
    {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"}};
```

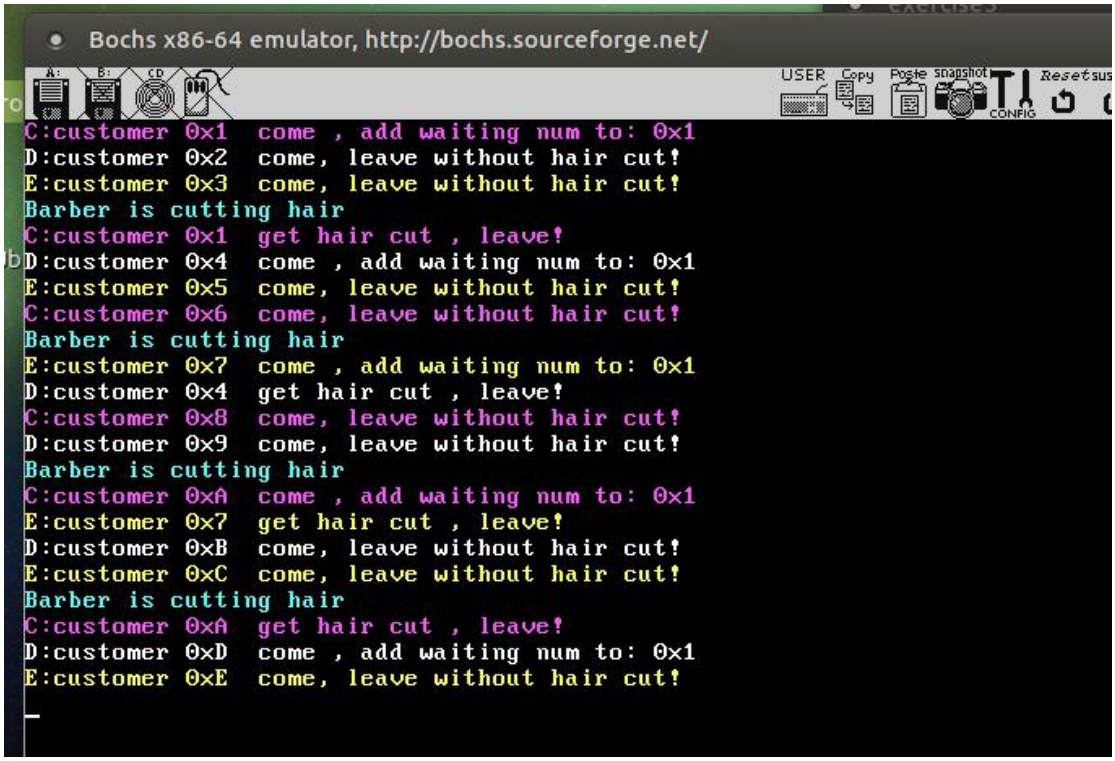
3、修改在 proc.h 的 NR_TASKS 的值，并给新增加的进程定义进程栈，修改栈的总大小

```
/* Number of tasks & procs */
#define NR_TASKS    1
#define NR_PROCS    5

/* 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)
```

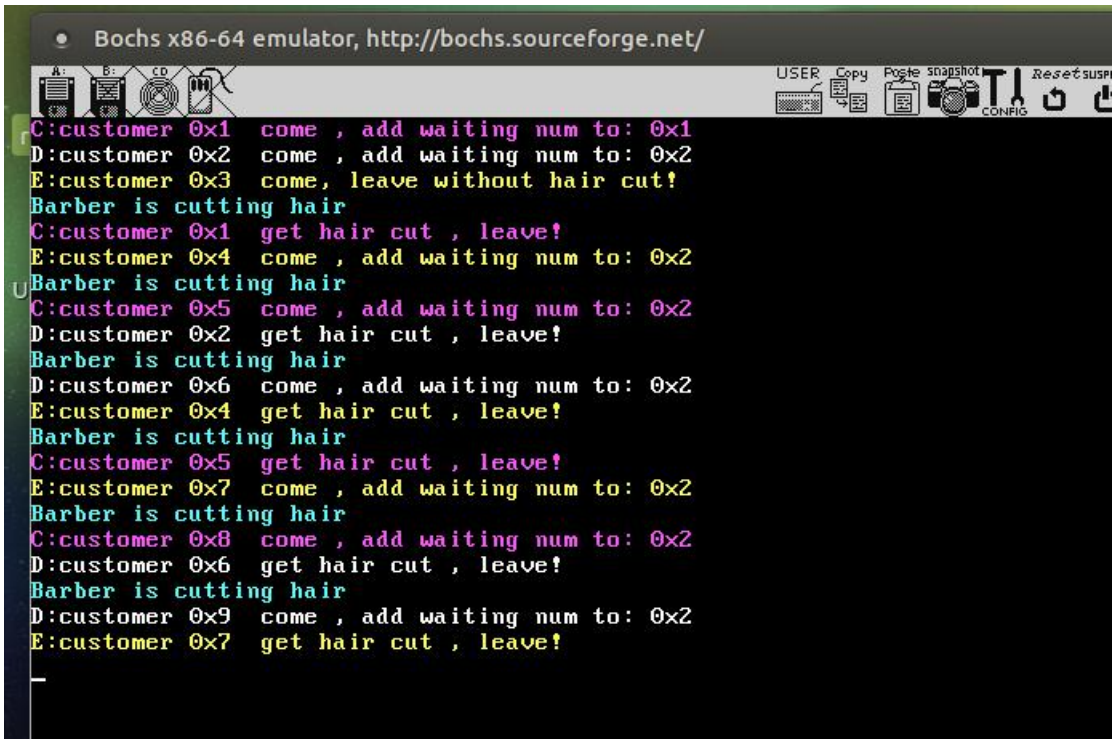
4、在 proto.h 中声明新增加的进程

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

Chais = 1



Chairs =2



Chairs = 3