操作系统第四次实验

151250060 黄潇

运行环境：

操作系统 32位 Linux Ubuntu 10.16

bochs x86 Emulator 2.6（装有bochs-sdl）

运行方式：

终端进入Makefile所在目录，输入make run运行程序

（请确保linux系统中有 /media/floppy 目录，用作a.img的挂载点）

参考代码

《Orange’s 一个操作系统的实现》第七章代码

修改内容

在随书光盘chapter7/o基础上修改

添加系统调用：

global.c:

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

proc.c:

```
PUBLIC void sys_process_sleep(int milli_seconds) {
    p_proc_ready->sleep = milli_seconds* HZ / 1000 + 1;
    schedule();
}

PUBLIC void sys_disp_str(char* str) {
    printf(str);
}

PUBLIC void sys_sem_p(semaphore* sem) {
    sem->value--;
    if (sem->value < 0) {
        sem->list[sem->len++] = p_proc_ready;
        p_proc_ready->is_ready = FALSE;
        // if barber process go sleep.
        if (p_proc_ready -> pid == 2) {
            my_disp_str("There is no customer, barber go sleeping.\n");
        }
        schedule();
    }
}

PUBLIC void sys_sem_v(semaphore* sem) {
    sem->value++;
    if (sem->value <= 0) {
        sem->list[0]->is_ready = TRUE;
        for (int i = 1; i < sem->len; ++i) {
            sem->list[i-1] = sem->list[i];
        }
        sem->len--;
    }
}
```

syscall.asm

```
_NR_process_sleep    equ 2
_NR_disp_str         equ 3
_NR_sem_p            equ 4
_NR_sem_v            equ 5

; 导出符号
global  get_ticks
global  write
global  process_sleep
global  my_disp_str
global  sem_p
global  sem_v
```

```
process_sleep:
    mov eax, _NR_process_sleep
    mov ebx, [esp + 4]
    int INT_VECTOR_SYS_CALL
    ret

my_disp_str:
    mov eax, _NR_disp_str
    mov ebx, [esp + 4]
    int INT_VECTOR_SYS_CALL
    ret

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

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

proto.h

```
/* 以下是系统调用相关 */

/* 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  void    sys_process_sleep(int mill_seconds);
PUBLIC  void    sys_disp_str(char* str);

/* 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  void    process_sleep(int mill_seconds);
PUBLIC  void    my_disp_str(char* str);
```

添加进程：
proc.h

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

/* stacks of tasks */
#define STACK_SIZE_TTY      0x5000
#define STACK_SIZE_TESTA    0x5000
#define STACK_SIZE_TESTB    0x5000
#define STACK_SIZE_TESTC    0x5000
#define STACK_SIZE_TESTD    0x5000
#define STACK_SIZE_TESTE    0x5000

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

proto.h

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

global.c

```
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"}};
```

main.c
理发师进程

```
/*======================================================================*
                            TestB
 *======================================================================*/
void TestB()
{
    int i = 0x1000;
    int customer;
    while(1){
        sem_p(&customers);
        sem_p(&mutex);
        customer = wait[0];
        for (int i = 1; i < waiting; ++i) {
            wait[i - 1] = wait[i];
        }
        waiting--;

        sem_v(&barbers);
        sem_v(&mutex);

        printf("Barber cut heir for Customer %x.\n", customer);
        milli_delay(50000);
        printf("Barber finish cutting. Customer %x leave.\n", customer);
    }
}
```

顾客进程（TestD、TestE相同）

```
/*======================================================================*
                              TestC
 *======================================================================*/
void TestC()
{
    int i = 0x2000;
    while(1){
        sem_p(&mutex);
        customerID++;
        printf("Customer %x come ", customerID);
        if (waiting < CHAIRS) {
            wait[waiting++] = customerID;
            printf("and wait. Waiting number: %x.\n", waiting);
            sem_v(&customers);
            sem_v(&mutex);
            sem_p(&barbers);
        } else {
            my_disp_str("and leave.\n");
            sem_v(&mutex);
        }
        milli_delay(30000);
    }
}
```

进程结构体修改——添加is_ready、sleep

```
typedef struct s_proc {
    STACK_FRAME regs;

    u16 ldt_sel;
    DESCRIPTOR ldts[LDT_SIZE];

        int ticks;
        int priority;

    u32 pid;
    char p_name[16];

    int nr_tty;

    int is_ready;
    int sleep;
}PROCESS;
```

修改进程调度算法

```
PUBLIC void schedule()
{
    for (int i = 0; i < NR_TASKS + NR_PROCS; ++i) {
        p_proc_ready++;
        if (p_proc_ready >= proc_table + NR_TASKS + NR_PROCS) {
            p_proc_ready = proc_table;
        }
        if (p_proc_ready -> is_ready && p_proc_ready -> sleep == 0) {
            break;
        }
    }
}
```

添加信号量定义：
proc.h

```
/* Definition of semaphore */
typedef struct {
    int value;
    PROCESS* list[MAX_QUEUE_SIZE];
    int len;
}semaphore;
```

添加全局变量：
global.h

```
EXTERN  int     waiting;
EXTERN  int     wait[8];
EXTERN  int     CHAIRS;
EXTERN  int     customerID;
EXTERN  semaphore   customers;
EXTERN  semaphore   barbers;
EXTERN  semaphore   mutex;
```

全局变量初始化
main.c

```
waiting = 0;
CHAIRS = 3;
customerID = 0;

// Initialize semaphore
customers.value = 0;
customers.len = 0;

barbers.value = 0;
barbers.len = 0;

mutex.value = 1;
mutex.len = 0;
```

更改输出颜色：
const.h

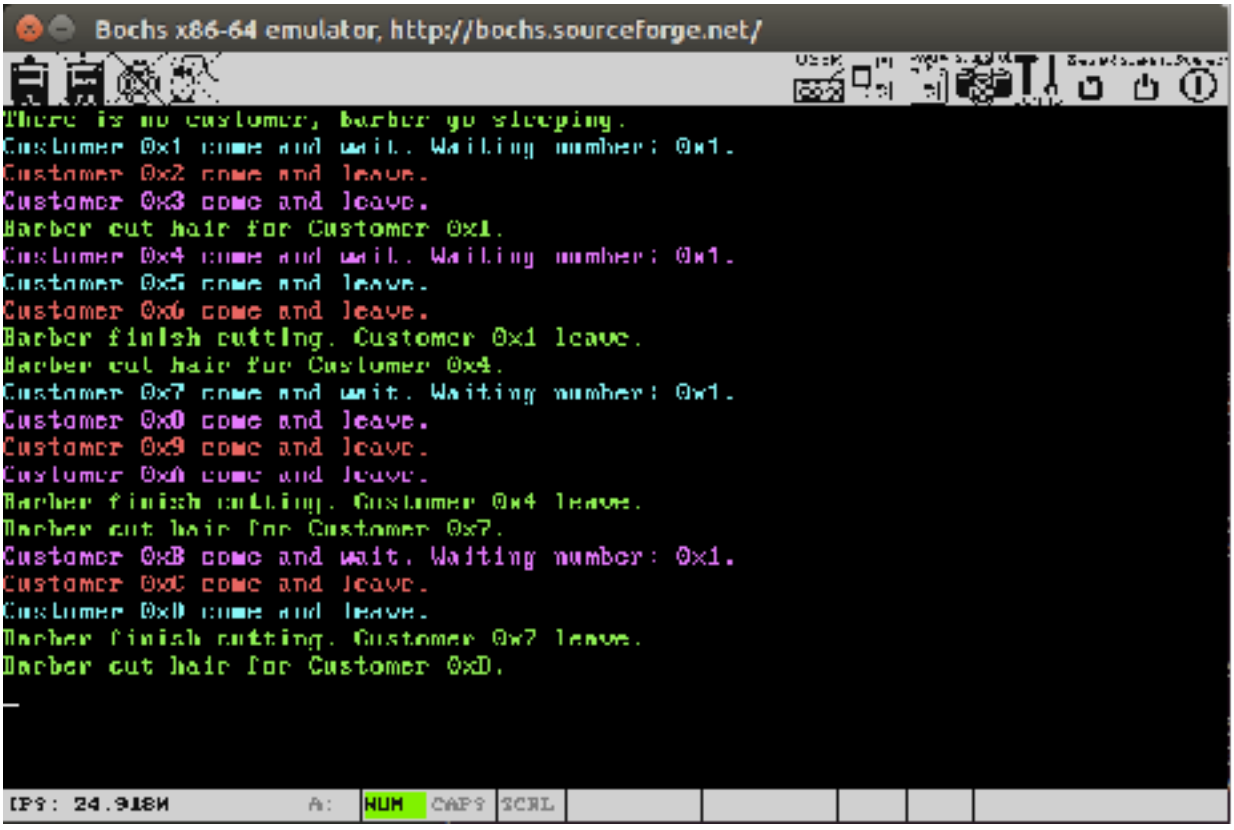
```
/* color */
/*
 * e.g. MAKE_COLOR(BLUE, RED)
 *      MAKE_COLOR(BLACK, RED) | BRIGHT
 *      MAKE_COLOR(BLACK, RED) | BRIGHT | FLASH
 */
#define BLACK   0x0     /* 0000 */
#define WHITE   0x7     /* 0111 */
#define BROWN   0x6     /* 0110 */
#define PINK    0x5     /* 0101 */
#define RED     0x4     /* 0100 */
#define CYAN    0x3     /* 0011 */
#define GREEN   0x2     /* 0010 */
#define BLUE    0x1     /* 0001 */
#define FLASH   0x80    /* 1000 0000 */
#define BRIGHT  0x08    /* 0000 1000 */
#define MAKE_COLOR(x,y) (x | y) /* MAKE_COLOR(Background,Foreground) */
```

console.c out_char

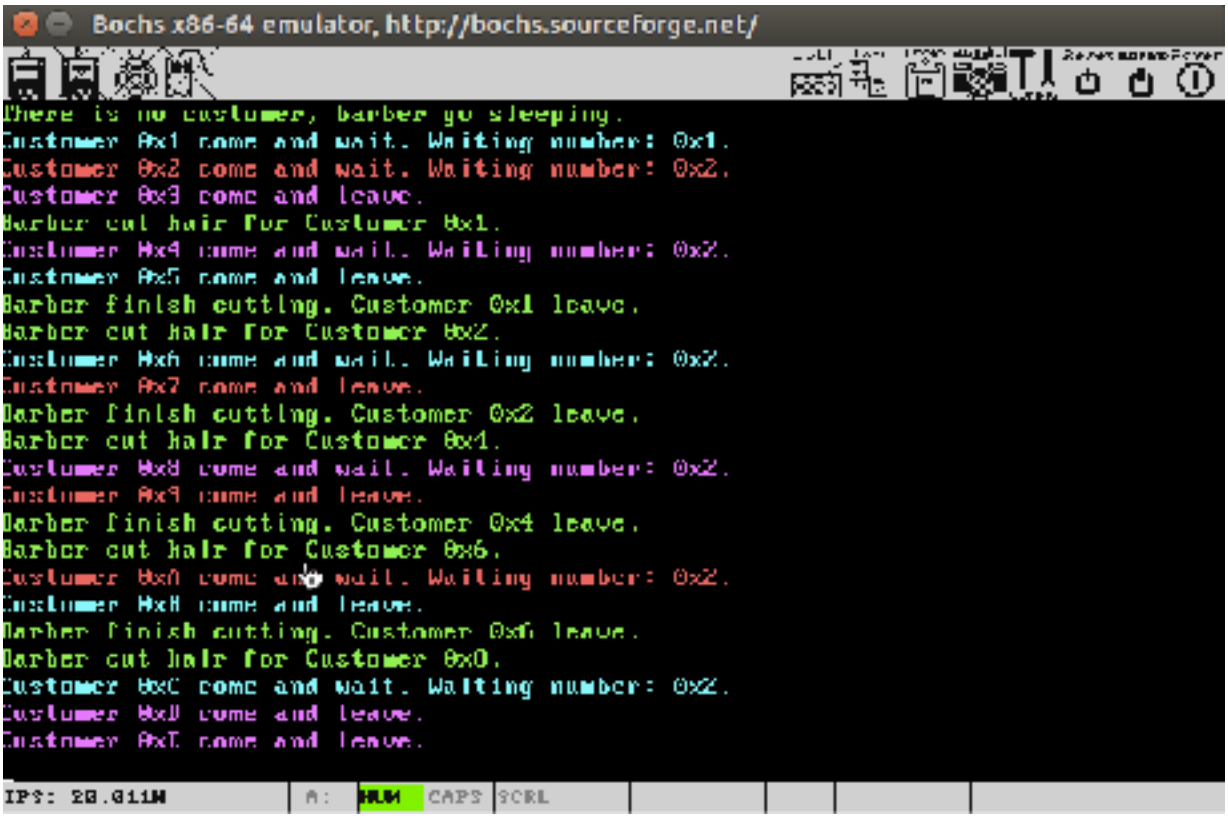
```
default:
    if (p_con->cursor <
        p_con->original_addr + p_con->v_mem_limit - 1) {
        *p_vmem++ = ch;
        // display different color by pid
        *p_vmem++ = BRIGHT | MAKE_COLOR(BLACK, p_proc_ready->pid);
        p_con->cursor++;
    }
    break;
}
```

输出结果

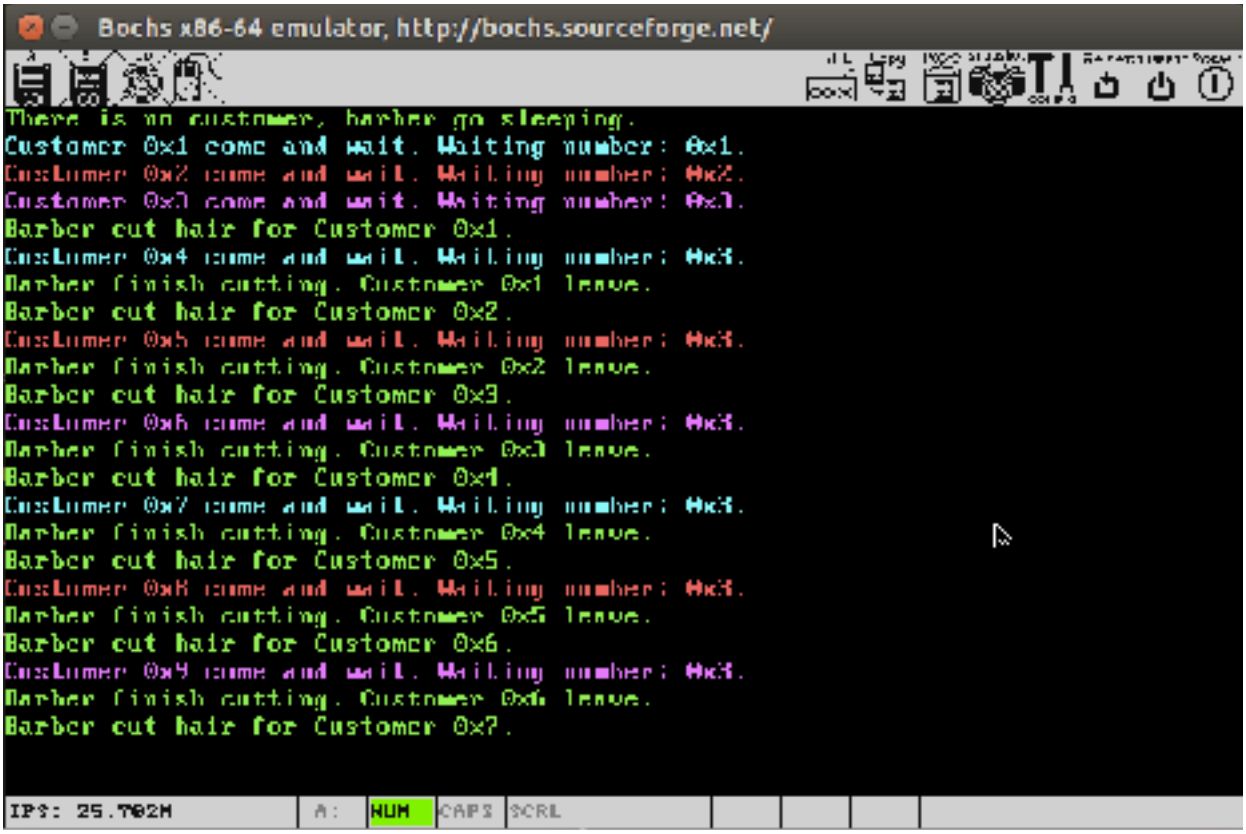
一把椅子：



两把椅子：


Bochs x86-64 emulator, http://bochs.sourceforge.net/
There is no customer, barber go sleeping.
Customer 0x1 come and wait. Waiting number: 0x1.
Customer 0x2 come and wait. Waiting number: 0x2.
Customer 0x3 come and leave.
Barber cut hair for Customer 0x1.
Customer 0x4 come and wait. Waiting number: 0x2.
Customer 0x5 come and leave.
Barber finish cutting. Customer 0x1 leave.
Barber cut hair for Customer 0x2.
Customer 0x6 come and wait. Waiting number: 0x2.
Customer 0x7 come and leave.
Barber finish cutting. Customer 0x2 leave.
Barber cut hair for Customer 0x4.
Customer 0x8 come and wait. Waiting number: 0x2.
Customer 0x9 come and leave.
Barber finish cutting. Customer 0x4 leave.
Barber cut hair for Customer 0x6.
Customer 0xA come and wait. Waiting number: 0x2.
Customer 0xB come and leave.
Barber finish cutting. Customer 0x6 leave.
Barber cut hair for Customer 0xA.
Customer 0xC come and wait. Waiting number: 0x2.
Customer 0xD come and leave.
Customer 0xE come and leave.
IPS: 28.611M
A:
NUM
CAPS
SCRL

三把椅子：


Bochs x86-64 emulator, http://bochs.sourceforge.net/
There is no customer, barber go sleeping.
Customer 0x1 come and wait. Waiting number: 0x1.
Customer 0x2 come and wait. Waiting number: 0x2.
Customer 0x3 come and wait. Waiting number: 0x3.
Barber cut hair for Customer 0x1.
Customer 0x4 come and wait. Waiting number: 0x3.
Barber finish cutting. Customer 0x1 leave.
Barber cut hair for Customer 0x2.
Customer 0x5 come and wait. Waiting number: 0x3.
Barber finish cutting. Customer 0x2 leave.
Barber cut hair for Customer 0x3.
Customer 0x6 come and wait. Waiting number: 0x3.
Barber finish cutting. Customer 0x3 leave.
Barber cut hair for Customer 0x4.
Customer 0x7 come and wait. Waiting number: 0x3.
Barber finish cutting. Customer 0x4 leave.
Barber cut hair for Customer 0x5.
Customer 0x8 come and wait. Waiting number: 0x3.
Barber finish cutting. Customer 0x5 leave.
Barber cut hair for Customer 0x6.
Customer 0x9 come and wait. Waiting number: 0x3.
Barber finish cutting. Customer 0x6 leave.
Barber cut hair for Customer 0x7.
IPS: 25.702M
A:
NUM
CAPS
SCRL