操作系统第四次实验说明文档

增加代码：

1. 在global.h中添加用于读者与写者进入临界区控制的信号量rmutex、wmutex以及用于实现写者优先以及读者优先的信号量r_lock、w_lock还有用于公平算法的信号量fair_lock，rmax则用于控制最大读者数,mode是读者优先还有写者优先的控制变量，readercount,writercount是用于读者写者计数的变量的。

```
EXTERN  int     ticks;
EXTERN char currentProc;
EXTERN Semaphore rmutex;
EXTERN Semaphore r_lock;
EXTERN Semaphore w_lock;
EXTERN Semaphore wmutex;
EXTERN Semaphore fair_lock;
EXTERN Semaphore rmax;
EXTERN int Mode;
EXTERN int readerCount;
EXTERN int writerCount;
```

2. 在proto.h中添加信号量结构体以及函数声明。

```
typedef struct Semaphore
{
    int value;
    int Q[Queue_SIZE];
    int phead;
    int ptail;
} Semaphore;
```

```
PUBLIC  void    sys_call();             /* int_handler */
PUBLIC  int     get_ticks();
PUBLIC void newdelay(int milli_sec);
PUBLIC void print(char *info);
PUBLIC void color_print(char *info, int color);
PUBLIC void P(Semaphore *s, int proc);
PUBLIC void V(Semaphore *s);
```

3. 在global.c中添加系统调用表以及进程表

```
PUBLIC TASK task_table[NR_TASKS] = {
    {ReaderA, STACK_SIZE_READERA, "ReaderA"},
    {ReaderB, STACK_SIZE_READERB, "ReaderB"},
    {ReaderC, STACK_SIZE_READERC, "ReaderC"},
    {WriterD, STACK_SIZE_WRITERD, "WriterD"},
    {WriterE, STACK_SIZE_WRITERE, "WriterE"},
    {F, STACK_SIZE_F, "F"},
};
PUBLIC  irq_handler     irq_table[NR_IRQ];

PUBLIC system_call sys_call_table[NR_SYS_CALL] = { sys_get_ticks,
                                                   sys_disp_str,
                                                   sys_disp_color_str,
                                                   sys_newdelay,
                                                   sys_P,
                                                   sys_V };
```

4. 在系统调用（syscall.asm）中增加打印函数print，自己写的调用延时函数newdelay以及PV操作对应的系统调用（在我理解这更像是通过系统调用将原语函数中的参数送入系统要处理的栈中），注意要设置对应的在系统调用表中的号以及导出的函数名称，注意要一一对应

```
_NR_get_ticks       equ 0 ; 要跟 global.c 中 sys_call_table 的定义相对应!
_NR_print           equ 1
_NR_color_print     equ 2
_NR_mydelay         equ 3
_NR_P               equ 4
_NR_V               equ 5
INT_VECTOR_SYS_CALL equ 0x90

; 导出符号
global  get_ticks
global  print
global  color_print
global  newdelay
global  P
global  V
```

```
; ====================================================================
;                              print
; ====================================================================
print:
    mov eax,_NR_print
    mov ebx, dword[esp + 4]
    int INT_VECTOR_SYS_CALL
    ret

; ====================================================================
;                              color_print
; ====================================================================
color_print:
    mov eax,_NR_color_print
    mov ebx, dword[esp + 4]
    mov ecx, dword[esp + 8]
    int INT_VECTOR_SYS_CALL
    ret
; ====================================================================
;                              new_delay
; ====================================================================
newdelay:
    mov eax,_NR_mydelay
    mov ebx, dword[esp+4]
    int INT_VECTOR_SYS_CALL
    ret
; ====================================================================
;                              P
; ====================================================================
P:
    mov eax,_NR_P
    mov ebx, dword[esp+4]
    mov ecx, dword[esp+8]
    int INT_VECTOR_SYS_CALL
    ret
```

```
; ====================================================================
;                              V
; ====================================================================
V:
    mov eax,_NR_V
    mov ebx, dword[esp+4]
    int INT_VECTOR_SYS_CALL
    ret
```

5. 在proc.c中添加（1）中对应系统调用的原语函数，并添加到global.c的系统调用表中去，这样会在syscall.asm运行时寻找到对应的原语函数

```c
/*======================================================================*
                           sys_disp_color_str
 *======================================================================*/
PUBLIC void sys_disp_color_str(char* info, int color)
{
    disp_color_str(info, color);
    if (disp_pos >= 2 * 80 * 25)
    {
        clearscreen();
    }
}
/*======================================================================*
                           sys_disp_str
 *======================================================================*/
PUBLIC void sys_disp_str(char* info)
{
    disp_str(info);
    if (disp_pos >= 2 * 80 * 25)
    {
        clearscreen();
    }
}
/*======================================================================*
                           sys_mydelay
 *======================================================================*/
PUBLIC void sys_newdelay(int milli_sec)
{
    p_proc_ready->ticks = milli_sec * HZ / 1000;
    while (1)
    {
        p_proc_ready = (p_proc_ready + 1 - proc_table) % NR_TASKS + proc_table;
        if (p_proc_ready->ticks <= 0 && p_proc_ready->blocked != 1)
            break;
    }
}
```

```
                            sys_P
 *======================================================================*/
PUBLIC void sys_P(Semaphore* s, int proc)
{
    s->value--;
    if (s->value < 0)
    {
        s->Q[s->phead] = proc;
        p_proc_ready->blocked = 1;
        while (1)
        {
            p_proc_ready = (p_proc_ready + 1 - proc_table) % NR_TASKS + proc_
            if (p_proc_ready->ticks <= 0 && p_proc_ready->blocked != 1)
                break;
        }//阻塞当前程序并寻找下一能运行程序
        s->phead = (s->phead + 1) % Queue_SIZE;
    }
}
/*======================================================================*
                            sys_V
 *======================================================================*/
PUBLIC void sys_V(Semaphore* s)
{
    s->value++;
    if (s->value <= 0)
    {
        int proc = s->Q[s->ptail];
        proc_table[proc].blocked = 0;
        p_proc_ready = proc_table + proc; //唤醒队尾程序
        s->ptail = (s->ptail + 1) % Queue_SIZE;
    }
}
```

6. 然后在main.c中添加对应的进程并且在对应的位置作好声明。写有关pv操作以及时间片的算法，其中设置这三个参数可以更改状态，max_reader是设置最大读者数，mode是读者优先还是写者优先（0/1），hungry是是否使用公平算法。

```
#define MAX_READER 3//最大允许读者数
#define MODE 1
#define HUNGRY 1
```

```
proc_table[0].ticks = proc_table[0].blocked = 0;
proc_table[1].ticks = proc_table[1].blocked = 0;
proc_table[2].ticks = proc_table[2].blocked = 0;
proc_table[3].ticks = proc_table[3].blocked = 0;
proc_table[4].ticks = proc_table[4].blocked = 0;
proc_table[5].ticks = proc_table[5].blocked = 0;
init();
//Init --End
k_reenter = 0;
ticks = 0;
p_proc_ready = proc_table;
/* 初始化 8253 PIT */
out_byte(TIMER_MODE, RATE_GENERATOR);
out_byte(TIMER0, (u8)(TIMER_FREQ / HZ));
out_byte(TIMER0, (u8)((TIMER_FREQ / HZ) >> 8));
put_irq_handler(CLOCK_IRQ, clock_handler); /* 设定时钟中断处理程序 */
enable_irq(CLOCK_IRQ);                     /* 让8259A可以接收时钟中断 */
restart();
while (1){ }
```

```
void init() {
    setS(&rmutex, 1);
    setS(&wmutex, 1);
    setS(&r_lock, 1);
    setS(&w_lock, 1);
    setS(&fair_lock, 1);
    setS(&rmax, MAX_READER);
    readerCount = writerCount = 0;
    Mode = MODE;
    clearscreen();
}
/*=============================================================================*
                                TestA
*=============================================================================*/
void ReaderA()
{
    protol('A', COLOR_A);
}

void ReaderB()
{
    protol('B', COLOR_B);
}

void ReaderC()
{
    protol('C', COLOR_C);
}
void WriterD()
{
    protol('D', COLOR_D);
}
```

```
void WriterE()
{
    protol('E', COLOR_E);
}
void protol(char name,int color) {
    if (color < 3) {
        while (1)
        {
            if (!HUNGRY)
                P(&fair_lock, color);
            if (Mode == 1)
                P(&rmutex, color);
            P(&r_lock, color);
            if (!readerCount)
                P(&wmutex, color);
            readerCount++;
            V(&r_lock);
            if (Mode == 1)
                V(&rmutex);
            if (!HUNGRY)
                V(&fair_lock);

            P(&rmax, color);
            char out1[20] = "Reader   Start!\n\0";
            char p1 = name+color;
            out1[7] = p1;
            switch (name) {
            case 'A':
                color_print(out1, RED);
                break;
            case 'B':
                color_print(out1, BLUE);
                break;
            case 'C':
                color_print(out1, COLOR1);
                break;
            default:
                break;
            }
            currentProc = name + color;
            char out2[20] = "  is Reading!\n\0";
            char p2 = name+color;
            out2[0] = p2;
            color_print(out2, COLOR2);
            switch (name) {
            case 'A':
                milli_delay(2 * TIMECOUNT);
                break;
            case 'B':
                milli_delay(3 * TIMECOUNT);
                break;
            case 'C':
                milli_delay(3 * TIMECOUNT);
                break;
            default:
                break;
            }
            char out3[20] = "Reader   End!\n\0";
            char p3 = name+color;
            out3[7] = p3;
            color_print(out3, COLOR4);
            V(&rmax);

            P(&r_lock, color);
            readerCount--;
```

```
            if (!readerCount)
            {
                V(&wmutex);
            }
            V(&r_lock);
        }
    }
    else {
        while (1)
        {
            if (!HUNGRY)
                P(&fair_lock, color);
            if (Mode == 1)
            {
                P(&w_lock, color);
                writerCount++;
                if (writerCount == 1)
                    P(&rmutex, color);
                V(&w_lock);
            }
            P(&wmutex, color);
            char out4[20] = "Writer   Start!\n\0";
            char p4 = name+color;
            out4[7] = p4;
            switch (name) {
            case 'D':
                color_print(out4, GREEN);
                break;
            case 'E':
                color_print(out4, COLOR5);
                break;
            default:
                break;
            }
            currentProc = name + color;
            char out5[20] = "  is Writing!\n\0";
            char p5= name + color;
            out5[0] = p5;
            color_print(out5, COLOR5);
            switch (name) {
            case 'D':
                milli_delay(3 * TIMECOUNT);
                break;
            case 'E':
                milli_delay(4 * TIMECOUNT);
                break;
            default:
                break;
            }
            char out6[20] = "Writer   End!\n\0";
            char p6 = name+color;
            out6[7] = p6;
            color_print(out6, GREEN);
            V(&wmutex);

            if (Mode == 1)
            {
                P(&w_lock, color);
                writerCount--;
                if (writerCount == 0)
                    V(&rmutex);
                V(&w_lock);
            }
            if (!HUNGRY)
                V(&fair_lock);
        }
```

```
    }
}

void F()
{
    while (1)
    {
        newdelay(1 * TIMECOUNT);
        int flag = judge();
        if(flag)
        {
            char out[20] = "readerCount:  \n\0";
            out[13] = '0' + readerCount; //TODO
            print(out);
        }
    }
}
int judge() {
    int flag=1;
    switch (currentProc) {
    case 'A':flag = 1;
    case 'B':flag = 1;
    case 'C':flag = 1;
    case 'D':flag = 0;
    default:break;
    }
    return flag;
}
```