实验四说明文档

任务1 添加系统调用

以添加打印字符串函数为例：

首先在syscall.asm中添加休眠函数的调用主体，ebx保存的是休眠的时间

```
print_str:
    mov ebx, [esp + 4]                  ;exb保存了指向字符串的指针
    mov eax, _NT_print_str
    int INT_VECTOR_SYS_CALL
    ret
```

然后再global.c的系统调用表上添加要实现的系统调用：

```
PUBLIC system_call sys_call_table[NR_SYS_CALL] = {sys_get_ticks,
sys_delay_milli_seconds, sys_disp_str, sys_p, sys_v};
```

然后再proc.c中实现该系统调用函数

```
PUBLIC void sys_print_str(char *str)
{
    disp_str(str);
}
```

其余添加系统调用过程类似

任务2 读者写者问题

主要的代码逻辑如下：

mode是一个全局变量，其中mode为0代表读者优先，mode为1代表写者优先

在读者优先模式下，读者进程如果先检查自己是不是第一个读者，如果是则获得写锁，让写者进程阻塞，然后获得读锁开始读，在退出时如果自己是最后一个读者，则释放写锁，让写者进程可以写。写者优先策略的思路也相同。

```
void read(int slices)
{
    if (mode == 0)
    { //读者优先

        atomicP(&rmutex);
        if (readCount == 0)
        {
```

```
            atomicP(&rw_mutex); //有进程在读的时候不让其它进程写
        }
        readCount++;
        atomicV(&rmutex);

        atomicP(&nr_readers);
        disp_read_start();


        for (int i = 0; i < slices; i++)
        {
            disp_reading();
            milli_delay(TIMESLICE);
        }

        disp_read_end();
        atomicV(&nr_readers);

        atomicP(&rmutex);
        readCount--;
        if (readCount == 0)
        {
            atomicV(&rw_mutex);
        }
        atomicV(&rmutex);
    }
    else if (mode == 1)
    {

        P(&r);
        P(&rmutex);
        if (readCount == 0)
        {
            P(&w);
        }
        readCount++;
        V(&rmutex);
        V(&r);
        //V(&queue);

        atomicP(&nr_readers);
        disp_read_start();

        //读操作消耗的时间片
        //milli_delay(slices * TIMESLICE);
        for (int i = 0; i < slices; i++)
        {
            disp_reading();
            milli_delay(TIMESLICE);
        }

        disp_read_end();
        atomicV(&nr_readers);

        atomicP(&rmutex);
        readCount--;
        if (readCount == 0)
        {
```

```
            atomicV(&w);
        }
        atomicV(&rmutex);
    }
}

void write(int slices)
{
    if (mode == 0)
    {
        //读者优先
        atomicP(&rw_mutex);
        writeCount++;
        disp_write_start();
        //milli_delay(slices * TIMESLICE);
        for (int i = 0; i < slices; i++)
        {
            disp_writing();
            milli_delay(TIMESLICE);
        }
        disp_write_end();
        writeCount--;
        atomicV(&rw_mutex);
    }
    else if (mode == 1)
    {
        P(&wmutex);
        if (writeCount == 0)
        {
            P(&r); //申请r锁
        }
        writeCount++;
        V(&wmutex);

        P(&w);
        disp_write_start();
        //milli_delay(slices * TIMESLICE);
        for (int i = 0; i < slices; i++)
        {
            disp_writing();
            milli_delay(TIMESLICE);
        }
        disp_write_end();
        V(&w);

        P(&wmutex);
        writeCount--;
        if (writeCount == 0)
        {
            V(&r);
        }
        V(&wmutex);
    }
}
```

其次为了让F进程每隔一段时间打印一次，在schedule()调度函数时我检查当前的ticks如果和上一次记录的lastticks相差超过一定时间，则强行将当前的运行进程切换为F进程。

任务3 生产者消费者问题

主要代码逻辑如下：

有四个信号量，cmutex控制剩余的空间数，mutex控制各个进程对仓库的互斥访问，sget1和sget2控制对两组消费者进程。

```
void produce1()
{
    atomicP(&cmutex);
    atomicP(&mutex);
    milli_delay(TIMESLICE*3);
    p_proc_ready->count++;
    atomicV(&sget1);
    atomicV(&mutex);
}

void produce2()
{
    atomicP(&cmutex);
    atomicP(&mutex);
    milli_delay(TIMESLICE*3);
    p_proc_ready->count++;
    atomicV(&sget2);
    atomicV(&mutex);
}

void consumer1()
{
    atomicP(&sget1);
    atomicP(&mutex);
    milli_delay(TIMESLICE*3);
    p_proc_ready->count++;
    atomicV(&mutex);
    atomicV(&cmutex);
}

void consumer2and3()
{
    atomicP(&sget2);
    atomicP(&mutex);
    milli_delay(TIMESLICE*3);
    p_proc_ready->count++;
    atomicV(&mutex);
    atomicV(&cmutex);
}
```