

实验八：进程同步机制

目录

一/二.实验目的与要求.....	1
三.实验方案.....	1
1.实验工具与环境（gcc+nasm+ld）.....	1
2.方案的思想.....	2
1.相关原理：.....	2
2.实验内容.....	2
3.实现方法：.....	3
3.程序流程：.....	3
4.程序关键模块：.....	3
四.实验过程与结果.....	5
1.实验过程：.....	5
2.运行结果：.....	5
3.遇到的问题及解决情况：.....	6
五.实验创新点.....	7
六.实验总结：.....	7
参考文献：.....	7

一/二.实验目的与要求

1. 设计信号量机制
2. 使用信号量机制解决一个操作系统问题
3. 使用信号量实现基本的同步机制

三.实验方案

1.实验工具与环境（gcc+nasm+ld）

nasm 编译汇编代码生成.o 文件，gcc 编译 C 模块生成.o 文件；ld 链接.o 文件生成.bin 文件；dd 将.bin 写入映像文件.img；vmware 上添加.img 文件到软盘后运行。

2.方案的思想

1.相关原理:

①关于信号量:

用于进程间传递信号的一个整数值。在信号量上只可进行三种操作，即初始化，递减和增加，并且这三种操作都是原子操作。

② 关于同步

合作进程在并发时，利用计数信号量，可以按规定的时序执行各自的操作，实现复杂的同步，确保进程并发的情况正确完成使命

③关于互斥:

即当一个进程在临界区访问共享数据时，其他进程不能进入该临界区访问任何共享资源的情形。

2.实验内容

(这次实验建立在上次实验的基础上，用户通过键盘输入来运行退出用户子程序。而子程序1就是这次实验我关于有限缓冲区的生产者消费者模型的处理。这里最主要讲的就是这个子程序。)

1. struct semaphore, 信号量的内容

信号量是以结构体存在的，其中包含了信号量的id，信号量的值，信号量是否被使用的标志，一个阻塞数组，阻塞队列大小。为了实现简单，我是使用数组来放置阻塞队列的。

2. do_p(int semid)原语，递减信号量。

使信号量的值减一。当信号量的值变为负数时，阻塞执行semwait的进程，否则进程继续执行。

3.do_v(int semid)原语，递增信号量。

使信号量的值加一。若值小于等于0，则被P操作阻塞的进程解除阻塞。

3.实现方法：

①do_p()原语实现方法：

如上描述一致。当阻塞一个进程时，用信号量的阻塞队列标记进程id，然后将进程进行阻塞处理。这是上个实验的内容。

②do_v()原语的方法：

如上描述一致。当解除进程阻塞时，要遵循FIFO的原则。

3.程序流程：

生产者消费者模型的过程

- 1.事先设置好信号量的初始值，这需要三个信号量，一个保证互斥，一个防止缓冲为空时的消费，一个防止缓冲溢出。
- 2.父进程创建子进程，然后分别成为生产者消费者。
- 3.生产者和消费者不断生产消费，并且对缓冲的控制要互斥。

4.程序关键模块：

1.do_p 原语：

```
void do_p(int semid)
{
    sem[semid].value -= 1;
    if(sem[semid].value < 0)
    {
        sem[semid].block_list[sem[semid].block_size] = running_proc;
        sem[semid].block_size += 1;
        do_wait();
    }
}
```

传入信号量 id，对信号量减一，value 是信号量的值。信号量小于 0 时，将进程加入阻塞队列并进行阻塞。

2.do_v 原语:

```
void do_v(int semid)
{
    sem[semid].value += 1;
    if(sem[semid].value <= 0)
    {
        status[sem[semid].block_list[0]] = 2;
        sem[semid].block_size -= 1;
        int i;
        for(i = 0; i < sem[semid].block_size; i++)
        {
            sem[semid].block_list[i] = sem[semid].block_list[i+1];
        }
    }
}
```

将信号量加一。当信号量小于等于 0 时，将第一个阻塞的进程设为就绪态。

4.信号量:

```
struct semaphore{
    ... int value;
    int block_size;
    ... int block_list[5];
    ... int used;
};

sem[1].value = 1; → → → → // 互斥
sem[2].value = 0; → → → → // 代表现在缓冲区里的东西
sem[3].value = 5; → → → → // 代表缓冲区大小为5
```

这里有个比较大的缺陷，没实现信号量初始化操作

5.子程序 1

```

if (pid > 0) //父进程·生成者
{
    while(1)
    {
        delay(50); //·生产速度
        P(s3);
        P(s1);
        append();
        V(s1);
        V(s2);
    }
}
else //子进程
{
    while(1)
    {
        P(s2);
        P(s1);
        take();
        V(s1);
        V(s3);
        delay(150); //·消费速度
    }
}

```

s1 信号量用于互斥

s2 信号量用于防止缓冲区空时的取操作

s3 信号量用于防止缓冲区溢出

四.实验过程与结果

1.实验过程：

【编译链接的命令】

与上次实验一致。

【磁盘与内存安排】

与上次实验一致。

2.运行结果：

1.当生产速度大于消费速度时：


```
int_21h_6:
→ mov cx, bx
→ push ecx
→ xor ecx, ecx
→ mov ecx, _do_p
→ call ecx
→ pop ecx
→ jmp int_21h_end
```

这个问题还是吃亏在了对C和汇编的相互调用不够了解上面。以前对于C函数的调用，我都是无参的调用，所以不知道如果传参数进去，到底应该是谁来清理压进去的参数。我一开始是以为由C函数处理，结果程序就死机了。还好这次实验内容较为简洁，我用排错法，发现问题出在了传参上面，而且传进去的参数没有问题。于是就把问题锁定在了pop上面，然后再和同学交流了一下汇编调用带参C函数的参数处理，就解决这个问题了。

五.实验创新点

六.实验总结：

这次实验我做的是有限缓冲区的生产者消费者模型，这个模型的解决方法书上是有伪代码的。所以说解决起来并不麻烦，最主要是把信号量的机制设计出来，可以实现同步和互斥就可以解决问题了。

这次实验我遇到的困难不算多，毕竟生产者消费者问题也不难。最大的问题就是上面我说到的汇编调用带参C函数问题，这个问题我以前总是尽量避免，因为自己本身就了解的不清楚，所以我以前汇编调用的所有C函数都是无参的。但是很明显，只有实现汇编调用带参C函数，才是真正的C和汇编的相互交流。代码虽然不多，但是，要在其中发现问题所在总是很困难，我这次就是用蠢方法一步一步的进行调试，不断实验，然后才找到的问题出现的地方。这个问题我一直避免，没想到最后还是不得不去解决。

参考文献：

1. 《80x86 汇编语言程序设计教程》（杨季文）
2. 《nasm 中文手册》（csdn）

3. 《一个操作系统的实现》 于渊
4. 《汇编语言（第三版）》 王爽