School of Data and Computer Science, Sun Yat-sen University

《操作系统实验》

实验七：进程同步机制

17341190 叶盛源

2019 年 6 月 24 日

Contents

中山大學 SUN YAT-SEN UNIVERSITY

操作系统实验报告

1 实验目的:

实现用于互斥和同步的计数信号量机制

2 实验要求:

在这个项目中，我们完善进程模型

1) 多个进程能够利用计数信号量机制实现临界区互斥
2) 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) 如果内核实现了信号量机制相关的系统调用，并在c库中封装相关的系统调用，那么，我们的c语言就也可以实现多进程同步的应用程序了。

   例如：利用进程控制操作，父进程f创建二个子进程s和d，大儿子进程s反复向父进程f祝福，小儿子进程d反复向父进程送水果(每次一个苹果或其他水果)，当二个子进程分别将一个祝福写到共享数据a和一个水果放进果盘后，父进程才去享受：从数组a收取出一个祝福和吃一个水果，如此反复进行。

3 实验方案:

3.1 实验环境：

1) 实验运行环境：Windows10
2) 虚拟机软件：VMware Function和DOSBox
3) TCC+Tasm+TLink 混合编译链接
4) NASM 编译器

3.2 实验工具：

1) 汇编语言：NASM、TASM
2) 文本编辑器：VScode、notepad++
3) 软盘操作工具：WinHex

3.3 实验原理和思想：

1) 信号量实现：

   i) 信号量：一个整数和一个指针组成的结构体，在内核可以定义若干个信号量，统一编号。

   ii) 内核实现do_p()原语，在c语言中用p(int sem_id)调用

   iii) 内核实现do_v()原语，在c语言中用v(int sem_id)调用

   iv) 内核实现do_getsem()原语，在c语言中用getsem(int)调用,参数为信号量的初值

   v) 内核实现do_freesem(int sem_id),在c语言中用freesem(int sem_id)调用

2) 信号量机制

   i) 信号量是内核实现的。每个信号量是一个结构体，包含两个数据域：数值count和阻塞队列头指针next。我们在内核定义一个信号量数组，同时，内核还要实现p操作和v操作。

   ii) 我们设置4个系统调用为用户使用信号量机制。

   iii) int SemGet(int value): 向内核申请一个内核可用信号量，并将其count域初始化为value，返值就是内核分配的一个可用信号量在数组的下标，如果没有可用的信号量，返值为－1。

4 实验过程：

1) 信号量结构

我们首先需要定义一个信号量的结构体，他包含了一个信号量资源数、一个阻塞的排队队列和一个使用位used还有两个指针用于排队队列构成循环数组，之后我们建立一个信号量队列，每个元素都是一个信号量结构体。代码如下：

```
typedef struct semaphoretype {
    int count;
    int blocked_pcb[nrpcb];
    int used, front, tail;
} semaphoretype;

semaphoretype semaphorequeue[nrsemaphore]; /* 定义信号量的数组 */
```

2）semaGet函数

这个函数是用来创建一个空闲信号量的，我们通过传入value作为初值赋值给信号量。我们遍历我们的信号量数组，然后找到一个空闲的（used位为0的）信号量块，然后初始化它的各个属性并将value值赋给它的count

```
/* 获取一个空闲的信号量并初始化它 */
int semaGet(int value) {
    int i = 0;
    while (semaphorequeue[i].used == 1 && i < nrsemaphore) { ++i; }
    if (i < nrsemaphore) {
        semaphorequeue[i].used = 1;
        semaphorequeue[i].count = value;
        semaphorequeue[i].front = 0;
        semaphorequeue[i].tail = 0;
        pcb_list[CurrentPCBno].regImg.AX = i;
        Switch();
        return i;
    }
    else {
        pcb_list[CurrentPCBno].regImg.AX = -1;
        Switch();
        return -1;
    }
}
```

3）semaBlock函数

这个函数是用来阻塞当前进程并在当前信号量后面排队，如果用户程序调用额这个函数，他就会就如BLOCKED状态，并被加入到信号量的阻塞队列的末尾，当循环队列已经满了就不会加入队列一直阻塞。

```
void semaBlock(int s) {
    pcb_list[CurrentPCBno].Process_Status = BLOCKED;
    if ((semaphorequeue[s].tail + 1) % nrpcb == semaphorequeue[s].front) {
        return;
    }
    semaphorequeue[s].blocked_pcb[semaphorequeue[s].tail] = CurrentPCBno;
    semaphorequeue[s].tail = (semaphorequeue[s].tail + 1) % nrpcb;
}

void semaWakeUp(int s) {
    int t;
    if (semaphorequeue[s].tail == semaphorequeue[s].front) {
        return;
    }
    t = semaphorequeue[s].blocked_pcb[semaphorequeue[s].front];
    pcb_list[t].Process_Status = READY;
    semaphorequeue[s].front = (semaphorequeue[s].front + 1) % nrpcb;
}
```

4) **semaFree函数**

这个函数是释放信号量块的，我们只需要把used位变成1即可。

5) **semaWakeup函数**

这个函数是当我们需要唤醒阻塞进程的时候调用的，我们会把循环队列中的进程取出一个，把它的状态变成ready，同时把他从循环队列中删去，代码如下：

```
void semaWakeUp(int s) {
    int t;
    if (semaphorequeue[s].tail == semaphorequeue[s].front) {
        return;
    }
    t = semaphorequeue[s].blocked_pcb[semaphorequeue[s].front];
    pcb_list[t].Process_Status = READY;
    semaphorequeue[s].front = (semaphorequeue[s].front + 1) % nrpcb;
}
```

6) **semaP函数**

这个函数是我们的P操作函数，我们在测试程序中通过调用这个函数来进行信号量的等待，我们先将信号量的资源数减1，然后判断资源数和0的大小关系，并选择是否在函数中使用semaBlock来阻塞进程。

```
/* 信号量的P操作，阻塞进程 */
void semaP(int s) {
    semaphorequeue[s].count--;
    if (semaphorequeue[s].count < 0) {
        semaBlock(s);
        Schedule();
    }
    Switch();
}
```

7) **semaV函数**

这个函数是我们的V操作函数，我们在测试程序中通过调用这个函数来进行释放资源，我们将信号量资源加一，然后判断和0的关系，并选择在函数中使用semaWakeup函数来取出一个进程恢复成ready状态，其他还在排队的进程则保持blocked状态。

```
void semaV(int s) {
    semaphorequeue[s].count++;
    if (semaphorequeue[s].count <= 0) {
        semaWakeUp(s);
        Schedule();
    }
    Switch();
}
```

8) 封装进int21h调用

和上一个实验一样我们要把这个实验实现的函数操作封装进int21h中，我们还是一样先判断ah的值再进行跳转，其中一个封装过程如下：

```
next11:
    cmp ah,12
    jnz next12
    jmp semaping
```

```
semaping:
    push ss
    push ax
    push bx
    push cx
    push dx
    push sp
    push bp
    push si
    push di
    push ds
    push es
    .386
    push fs
    push gs
    .8086

    mov ax,cs
    mov ds, ax
    mov es, ax

    call near ptr _Save_Process
    mov bx,ax
    push bx
    call near ptr _semaP
    pop bx
    iret
```

5 实验测试和结果：

接下来我们需要按照老师的要求编写测试文件来测试我们的信号量操作编写正确性。

我们要利用实验7实现的fork操作来创建分支子进程。我们先用一个父进程fork'出一个子进程，同时这个子进程再fork出一个子进程，两个子进程一个不断的放水果，一个不断的给父亲祝福，没进行一个操作，就会利用semaV给信号量的资源加1，然后父进程用semaP等待资源，如果资源数有两个了，他就可以执行并输出一句感谢的话。用这个小程序就可以检验我们的信号量部分实现是否正确了。最后我们把他们链接起来变成一个.com文件当作用户程序运行，就可以实现测试目的。代码如下：

```
main() {
    int s, tmp;
    s = semaGet(0);
    print("\r\nUser: forking...\r\n");
    tmp = fork();
    if(tmp) {
        while(1) {
            semaP(s);
            semaP(s);
            if(be_change) {
                print(words);
                be_change = 0;
            }
            print("Father enjoy the fruit ");
            printInt(fruit_disk);
            print("\r\n");
            fruit_disk = 0;
        }
    }
```

Figure 1: 父进程路线

```
else {
    print("User: forking again...\r\n");
    tmp = fork();
    if(tmp) {
        while(1) {
            be_change = 1;
            write("Father will live one year after anther forever!\r\n");
            semaV(s);
            delay();
        }
    }
    else {
        while(1) {
            putfruit();
            semaV(s);
            delay();
        }
    }
```

Figure 2: 子进程路线

我们需要在主程序中编写测试指令，我们直接利用上一个实验实验7的测试函数和测试指令来时用，当我们输入-test指令的时候我们就会启动测试程序并输出一些话，实验结果如下：

```
/* 测试程序默认放在第一个PCB块中 */
void run_testPro_EX7()
{
    sector_number=7;
    RunTestProm(Segment,sector_number);
    Segment+=0x1000;
    Program_Num=1;
    pcb_list[1].Used = 1;
}
```

Figure 3: 编写测试指令1

```
else if(com[0]=='-'&&com[1]=='t'
{
    cls();
    run_testPro_EX7();
    Delay2();
    cls();
}
```

Figure 4: 编写测试指令2

最后我们就可以在内核中调用测试指令来查看运行结果，结果如下图。我们可以看到当子进程在果篮中放了一个水果并说了一句祝福后，父进程就会拿走果篮中的水果并说出一句感谢的话语。这样符合我们测试程序设计的目标，测试效果图如下：

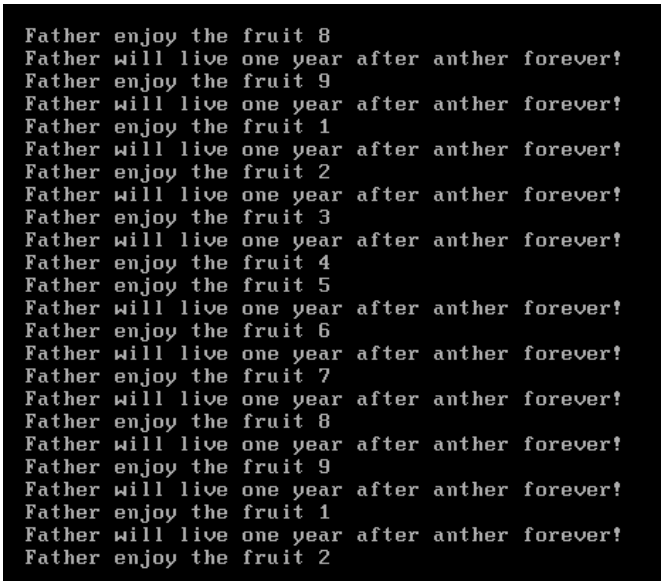

Figure 5: 测试结果

6 实验心得和总结：

本次实验是要实现我们在理论课上学到了信号量相关知识，我们需要建立一个信号量，来实现共享资源的互斥问题，也可以解决我们操作系统经常遇到的RC问题。

我们通过按照课本上的设计方法，设计了信号量P操作和V操作，他们一个是获取资源一个是释放资源，我们需要为每个信号量建立一个队列，阻塞或者释放上面的进程，来实现等待资源的这一个过程。

最后我们还按照老师的思路设计了一个用户程序，这个用户程序通过使用PV操作，和上一次实验实现的fork创建子进程的方法，来建立三个不同的进程，如果是子进程就一直V操作增加资源，如果是父进程就每次用两次P操作来取走资源，这样就可以模拟一个父亲取走水果，儿子送上祝福和水果的过程，也可以检验我们的信号量是否设计正确。有了实验七的基础，这次实验的难度

并不是特别大。在汇编代码部中，semaV、semaP、semaGet、semaFree这四个操作和实验七中的fork、wait、exit这三个操作十分相似，均是保护现场，将进程的上下文存入进程控制块中，再进入内核进行相关操作。

这次实验难度不大，但也学到了很多东西，接近期末了，我的操作系统实验项目应该就到此为止了，这个学期学到了很多东西，并自己动手设计出了一个最简单的操作系统模型，虽然还有很多东西需要我们自己动手去加上去，还不够完美，希望以后能有时间继续学习这门课程相关的内容，完善自己设计的原型操作系统。感觉助教和老师一个学期的付出，谢谢！希望自己继续努力，继续学好操作系统这门课程。

7 参考文献:

1) nasm手册

2) https://blog.csdn.net/longintchar/article/details/79511747

3) https://blog.csdn.net/cielozhang/article/details/6171783/

4) https://blog.csdn.net/lulipeng_cpp/article/details/8161982

5) https://www.cnblogs.com/alwaysking/p/7789282.html

6) https://blog.csdn.net/cielozhang/article/details/6171783

7) https://blog.csdn.net/lulipeng_cpp/article/details/8161982 https://stackoverflow.com/questions/12882342/override-default-int-9h