实验八报告

【个人信息】

院系：数据科学与计算机学院　　专业、年级：17 级计算机科学与技术
学号：17341097　　姓名：廖永滨　　指导教师：凌应标

【实验目的】

1、 保留实验 7 中的五状态进程模型，从而使进程可以分工合 作，并发运行。
2、 通过信号量实现进程同步机制

【实验要求】

1、 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。

2、 编写一个子程序验证信号量是否能够使用。此处以课件中的参考例子为主：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。

【实验方案】

1. 实验工具：

Notepad++：编写程序时使用的编辑器；
Subline：可以以 16 进制的方式打开并编辑任意文件；
TAMS 汇编工具：可以将汇编代码编译成对应的二进制代码；
NAMS 汇编工具：可以将汇编代码编译成对应的二进制代码；
TCC 编译器：可以将 c 代码编译成对应的二进制代码；
TLINK 链接器：将多个.obj 文件链接成.com 文件
VmWare 虚拟机：创建裸机环境，生成虚拟磁盘

2. 实验基本操作方法：

a. 使用 TCC 编译命令：tcc -mt -c - o cfile.obj cfile.c >ccmsg.txt
b. TASM 汇编命令：tasm afile.asm afile.obj > amsg.txt
c. 链接命令：tlink /3 /t cfile.obj afile.obj , final.com
d. NASM 汇编指令：NASM afile.asm

3. 实验原理

中断机制基础知识，中断程序的编写规范。

【实验过程】

1、实现的内容：

前一个原型中，操作系统可以用 do_fork，wait 等等语句进程解决多道程序技术并合作。但是这些进程模型还比较简单，进程之间虽然能互相往来，能进行合作。在实际的应用开发中，容易出现各种问题，合作不够到位，而且不好拓展，容易出现死锁等等大问题。如果设置好信号量这一元素，那么软件工程意义上更有优势。在这个项目中，我们完善进程模型，加入了信号量。

这个实验有个难点在于设计合作程序，我用了 PPT 上那个递苹果的例子编程，发现能解释好信号量这一元素，就没有进行创新。、本次实验是在实验七的基础上进行的，加入了计数信号量机制,实现了 do_p()、do_v()、do_getsem()、do_freesem(int sem_id)原语，并将这些原语封装进 C 库中，供用户程序使用。

主要加入的内容有：信号量的结构 semaphoretype，信号量数组 semaphorequeue，向内核申请一个内核可用信号量函数 semaGet(int value)，即实现 do_getsem()原语；释放信号量函数 semaFree(int s)，即实现 do_freesem(int sem_id)原语；实现将当前进程阻塞并放入信号量 s 的阻塞队列函数 semaBlock(int s)；实现唤醒信号量 s 的阻塞队列中的一个进程函数 semaWakeUp(int s)；实现 do_P()原语的函数 semaP(int s)；实现 do_V()原语的函数 semaV()；实现初始化信号量队列的函数 initsema()。semageting 服务、semafreeing 服务、semaping 服务、semaving 服务。

实现效果：

```
Father will live one year after anther forever!
Father enjoy the fruit 9
Father will live one year after anther forever!
Father enjoy the fruit 1
Father will live one year after anther forever!
Father enjoy the fruit 7
Father will live one year after anther forever!
Father enjoy the fruit 4
Father will live one year after anther forever!
Father enjoy the fruit 6
Father will live one year after anther forever!
Father enjoy the fruit 2
Father will live one year after anther forever!
Father enjoy the fruit 4
Father will live one year after anther forever!
Father enjoy the fruit 8
Father will live one year after anther forever!
Father enjoy the fruit 9
Father will live one year after anther forever!
Father enjoy the fruit 7
Father will live one year after anther forever!
Father enjoy the fruit 5
Father will live one year after anther forever!
Father enjoy the fruit 5
```

输入指令 t

2、原理：

（1） 进程基本的的同步机制

即用于互斥和同步的计数信号量机制，在这个项目中，我们完善进程模型。多个进程能够利用计数信号量机制实现临界区互斥。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

（2） 信号量机制

即计数信号量机制，信号量是指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。内核实现do_p()原语，在c语言中用p(int sem_id)调用；内核实现do_v()原语，在c语言中用v(int sem_id)调用；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；内核实现do_freesem(int sem_id),在c语言中用freesem(int sem_id)调用。

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。我们设置4个系统调用为用户使用信号量机制。函数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。

3、信号量相关模块说明：

（1）信号量结构：

```
*/
typedef struct semaphoretype {
    int count;
    int blocked_pcb[10];
    int used, front, tail;
} semaphoretype;
```

一个整数count，表示资源量；

一个数组，表示阻塞队列，一个循环队列；

标志位used，表示是否使用过；

队头front、队尾tail。

（2）信号量相关算法：

semaGet(int value)：申请信号量。在信号量队列中找到一个未使用的信号量，完成初始化操作后，将信号量下标返回。

semaFree(int s)：释放信号量。根据传递的参数决定释放的信号量。

semaP(int s)：P操作。对信号量的count--，如果小于0则将当前进程放到阻塞队列末尾，并重新调度。

semaV(int s)：V操作。对信号量的count++，如果小于等于0，则唤醒阻塞队列里面的第一个进程。

semaWakeUp(int s)：唤醒信号量s的阻塞队列中的一个进程。

semaBlock(int s)：将当前进程阻塞并放入信号量s的阻塞队列中。

4、代码解析（实现方法）

(1)MyOS.asm：

基本无变化。

(2)kliba.asm：

此处为中断的核心所在，在实验 7 的基础上，新增了很多过程，比如 semaving，semaping，semafreein,semageting 等等。

Semaving：

```
semaving:
    .386
    push ss
    push gs
    push fs
    push es
    push ds
    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax
    mov ax,cs
    mov ds, ax
    mov es, ax
    call _save_PCB
    mov bx,ax
    push bx
    call near ptr _semaV
    pop bx
    iret
```

Semaping:

```
semaping:
    .386
    push ss
    push gs
    push fs
    push es
    push ds
    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax
    mov ax,cs
    mov ds, ax
    mov es, ax
    call _save_PCB
    mov bx,ax
    push bx
    call near ptr _semaP
    pop bx
    iret
```

Semafreeing:

```
semafreeing:
    .386
    push ss
    push gs
    push fs
    push es
    push ds
    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax
    mov ax,cs
    mov ds, ax
    mov es, ax
    call _save_PCB
    mov bx,ax
    push bx
    call near ptr _semaFree
    pop bx
    iret
```

Semageting:

```
semageting:
    .386
    push ss
    push gs
    push fs
    push es
    push ds
    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax
    mov ax,cs
    mov ds, ax
    mov es, ax
    call near ptr _save_PCB
    mov bx,ax
    push bx
    call near ptr _semaGet
    pop bx
    iret
```

(3)kernal.c ：

新增的信号量相关的一系列函数。

```
246 int semaGet(int value) {
247     int i = 0;
248     while (semaphorequeue[i].used == 1 && i < 10) { ++i; }
249     if (i < 10) {
250         semaphorequeue[i].used = 1;
251         semaphorequeue[i].count = value;
252         semaphorequeue[i].front = 0;
253         semaphorequeue[i].tail = 0;
254         PCB_LIST[current_process_number].regs.ax = i;
255         PCB_Restore();
256         return i;
257     }
258     else {
259         PCB_LIST[current_process_number].regs.ax = -1;
260         PCB_Restore();
261         return -1;
262     }
263 }
264
```

```
265 void semaFree(int s) {
266     semaphorequeue[s].used = 0;
267 }
268
269 void semaBlock(int s) {
270     PCB_LIST[current_process_number].status = PCB_BLOCKED;
271     if ((semaphorequeue[s].tail + 1) % 10 == semaphorequeue[s].front) {
272         print("kernal: too many blocked processes\r\n");
273         return;
274     }
275     semaphorequeue[s].blocked_pcb[semaphorequeue[s].tail] = current_process_number;
276     semaphorequeue[s].tail = (semaphorequeue[s].tail + 1) % 10;
277 }
278
279 void semaWakeUp(int s) {
280     int t;
281     if (semaphorequeue[s].tail == semaphorequeue[s].front) {
282         print("No blocked process to wake up\r\n");
283         return;
284     }
285     t = semaphorequeue[s].blocked_pcb[semaphorequeue[s].front];
286     PCB_LIST[t].status = PCB_READY;
287     semaphorequeue[s].front = (semaphorequeue[s].front + 1) % 10;
288 }
289
```

```
290 void semaP(int s) {
291     semaphorequeue[s].count--;
292     if (semaphorequeue[s].count < 0) {
293         semaBlock(s);
294         schedule();
295     }
296     PCB_Restore();
297 }
298
299 void semaV(int s) {
300     semaphorequeue[s].count++;
301     if (semaphorequeue[s].count <= 0) {
302         semaWakeUp(s);
303         schedule();
304     }
305     PCB_Restore();
306 }
307
308 void initsema() {
309     int i;
310     for (i = 0; i < 10; ++i) {
311         semaphorequeue[i].used = 0;
312         semaphorequeue[i].count = 0;
313         semaphorequeue[i].front = 0;
314         semaphorequeue[i].tail = 0;
315     }
316 }
317
```

(4) user.c 等等 ：

由于实验要求我们设计一个能体现进程合作的子程序，按 PPT 上说的做就行。

【实验心得】

这次实验，最大的收获无异于对进程模型的进一步了解与合作方式的实现，以及信号量的使用与理论知识。在实验七的基础之上，实验八的完成算是简单了不少，毕竟知道了大概该改哪些地方。再加上网上有不少相关的资料，所以完成起来不少很复杂。

总的来说，通过这次实验，我对信号量有了更加深刻的理解。

首先，什么是信号量呢？理论课上讲了无数次，那就是为了防止出现因多个进程同时访问一个共享资源而引发的一系列问题而采取的信号量机制。在任一时刻只能有一个执行进程访问代码的临界区域。我们使用信号量来实现互斥，防止多个进程因同时访问共享数据而引发 RC 问题。

子程序就很好体现了这一点。反正就是一种规则，让大家能有序完成任务的规则。更何况这次实验子程序 PPT 已经给了，而且网上也有对应的子程序的 com 代码，直接搬过来就完事了。

所以事实上需要完成的还是信号量相关的那些函数。之中还有一些小问题：比如如何实现信号量阻塞队列、如何唤醒信号量的阻塞队列中的一个进程、如何将当前进程阻塞并放入信号量 s 的阻塞队列中。但这些问题都可以逐个击破。

不得不说，debug 依旧十分困难。这学期的操作系统实验做的头皮发麻，最后发现理论课知识并没有得到太多的巩固，感觉这门实验课还是需要进一步改进与完善，现在这种模式太硬核了。

【代码清单】

```
Project8
    --操作系统
        --KERNAL.C
        --KLIBA.ASM
        --MYOS.ASM
    --工具包
        --DOSBox.exe（只是快捷方式）
        --mydoc.conf  (DOSBOX 的配置文件)
        --nasm.exe
        --nasmpath.bat
        --TASM.EXE
        --TCC.EXE
        --TLINK.EXT
```

--软盘文件
    --实验 8.flp
--引导程序
    --loading.asm
    --loading.com
--用户程序
    --user.asm
    --user_lib.asm
    --user_lib.h
    --userc.c
    --user.com
    --userpro1.com
    --userpro2.com
    --userpro3.com
    --userpro4.com