操作系统原理实验

实验七

信号量与动态内存管理

姓名： 吴侃

学号： 14348134

班别： 2014 级计算机系一班

日期： 2016.04.30 – 2016.05.04

目录

零、特色先览

本次实验的亮点包括:

1. 信号量的实现 semWait(P 操作)， semWait(V 操作)
2. 信号量的手动与自动释放机制
3. 动态内存管理（内核为用户程序申请释放内存空间， 用户程序内部申请释放内存资源（实现了 C++的 new 和 delete））
4. 完善了 C++实现中断程序的机制

一、实验目的

实验信号量和内存管理

二、实验要求

1. 实现信号量并测试
2. 实现动态内存管理

三、实验环境与工具

（一）实验环境

物理机操作系统: Arch Linux 4.5.1-1

调试使用虚拟机: qemu-system-i386, bochs

虚拟机软件: VMware Workstation 12 Pro

虚拟机配置: CPU: i7-4702MQ @ 2.20GHz，使用单核单线程

内存:4 MB

硬盘:32 MB

（二）实验工具

编辑器: Vim 7.4

汇编工具: NASM 2.11.08

C++编译器: g++ 5.3.0

链接工具: GNU ld 2.26.0.20160302

构建工具: GNU Make 4.1

调试工具: Bochs x86 Emulator 2.6.8

虚拟机: qemu-system-i386

VMWare Workstation 12 Pro

合并文件: dd

四、实验方案

(1) 信号量的实现

信号量操作分为信号量的创建 semCreate，信号量的删除 semDel, P 操作 semWait, V 操作 semSignal.

信号量的存储：

在内核中使用一个信号量结构体 sem 数组， 其中 sem 的属性包括该信号量是否被使用，flag 值（保证该信号量在某一个时间段只被一个进程修改），count(信号量值)，next(阻塞链表的首进程 id), runid（创建该信号量的进程 id，用于自动释放信号量）

原子操作：

保证该操作被会被中断，这里使用 lock xchg 交换信号量的 flag 值。交换方式为：令一个变量 ax = 1, lock xchg [flag] ax，若最终结果 ax 为 0，说明可以操作信号量，同时 flag 被设置为 1. 若最终结果 ax 为 1，说明该信号量正在被操作，因此本进程要通过循环进行等待，直到可操作（ax = 0）为止。lock xchg 能保证该操作不被中断，并且一次只有一个进程修改该信号量。

阻塞队列：

每个信号量都有个 next 值，这是它阻塞队列入队最先的进程的 id 号。在 PCB 表里加一项 BLOCK_NEXT，指向下一个被阻塞的进程。入队时， 将进程加入链表末尾；出队时， 从链表头取一个进程。这里假定一个进程最多被一个信号量阻塞。

semCreate:

使用遍历方式找一个可用的信号量，设置信号量初值等数据，标记可用，返回信号量 id.

semWait:

首先检查是否可以修改信号量，若不能修改，则等待。若可修改，将信号量减一。这时若信号量为负数，将该进程阻塞，并加入该信号量对应的阻塞队列，然后 Schedule，即切换下一进程。

semSignal:

首先检查是否可以修改信号量，若不能修改，则等待。若可修改，将信号量加 1，若此时信号量小于等于零，说明之前有进程被阻塞。于是从该信号量的阻塞队列取出阻塞最久的进程，设置该进程为运行态。

semWait 和 semSignal 在修改信号量的过程中，flag 为 1，其它进程无法进入修改过程。当修改过程完毕，flag 被设置为 0.

(2) **内存管理**

基本描述:

使用一个定长数组实现链表结构，每个链表节点表示该区域最大连续可用空间，初始化时，只有一个有效节点，即可用内存地址的起始点

和终止点，这个区间内的内存可用并且连续。

内存申请：

使用最佳适应方法，每次找到最小并且满足申请空间大小的分区，从低地址开始占用需要的内存，返回内存地址。

内存释放：

找到要释放的内存本来的位置，查看和前后节点是否连续，决定是否需要进行节点合并操作。

单位：

对于内核申请内存给用户程序，使用段为单位。

对于用户程序内部申请内存（new 和 delete），以字节为单位。

(3) **系统调用**

将信号量的操作做成中断 25h, ah 决定功能号

将内存管理作为中断 23h

用户程序操作信号量或进行内存管理时，调用对应中断。

(4) **C++实现中断程序的改进**

g++编译出的二进制文件，其实是 32 位的，当加入-m16 编译参数时，

每条指令（push 和 pop）除外，在前面基本都加了 0x66，并且 g++对局部变量的栈偏移地址也是按 32 位的计算。但 push 和 pop 是根据当前运行模式决定使用 2 字节（16 位）还是 4 字节(32 位)。

我定义了三个 C++宏，CPP_INT_START，CPP_INT_END，CPP_INT_LEAVE，这些宏将放在中断程序的首尾。当中断程序不含局部变量时，组合为 START－END；当含局部变量时，组合为 START－LEAVE.

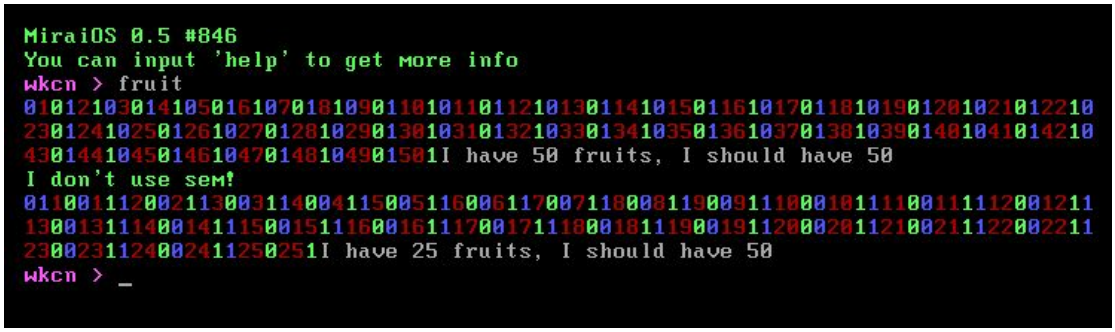
由于 iret 的指令为 0xcf，因此在内嵌汇编里使用.byte 0xcf (使用 iret 会导致编译出指令 0x66cf)

在每次进入中断程序时，将 ds 设为 cs 的值，确保内部变量可用；退出时还原 ds 值。

五、实验操作

信号量：

输入命令 fruit



两兄弟分别给父亲 50 个水果，蓝色数字表示进入计数过程的线程标识，绿色表示退出计数过程的线程标识，红色表示 count 值。可以看到使用信号量与不使用信号量的结果对比，只有使用信号量时计数正

确。

```
11 __attribute__((regparm(1)))
12 void * GiveFruit(void *p){
13     for (int i = 0;i < total / 2;++i){
14         if(useSem)semWait(sid);
15         PrintNum(*(uint16_t*)(p), LBLUE);// 进入
16         uint16_t old;
17         old = count;
18         for (uint16_t w=0;w<0xFFFF;++w){
19             for (uint16_t ww=0;ww<0xFF;++ww){
20                 for (uint16_t www=0;www<0x1;++www){
21                 }
22             }
23         }
24         count = old + 1;
25         PrintNum(count,RED); // 数字
26         PrintNum(*(uint16_t*)(p),LGREEN); // 出来
27         if(useSem)semSignal(sid);
28     }
29     return 0;
30 }
31
```

使用中间变量 old 和等待语句，使结果对比更明显

可以看见进程被阻塞

```
wkcn > top
 There are 4 Progresses :-)
 PID Name         PR  Size    SEG     CS      IP      Parent  State
 0   Mirai-Shell  0   0       0x0000  0xEA60  0x25B2  0       Running
 1   FRUIT.COM    0   11055   0x4170  0x4170  0x1B42  0       Blocked
 2   FRUIT.COM    0   11055   0x4433  0x0000  0xA07F  1       Suspend
 3   FRUIT.COM    0   11055   0x4443  0x0000  0xA07F  1       Blocked
wkcn > _
```

中途查看进程状态，可以看见一个线程（3）被阻塞，其中父进程被阻塞是因为其调用了 thread_join，在等待它申请的线程结束。

统计字母数量：

这里使用线程进行统计。

```
wkcn > letter
#Alpha of Str: 27
```

可知，

char str[80]="129djwqhdsajd128dw9i39ie93i8494urjoiew98kdkd";

共有 27 个字母。

内存管理：

内核申请内存：

创建 9 个进程，输入 top

```
wkcn > top
 There are 9 Progresses :-)
 PID Name         PR  Size    SEG     CS      IP      Parent  State
 0   Mirai-Shell  0   0       0x0000  0x0000  0xA94D  0       Running
 1   WKCN1.COM    0   512     0x4170  0x4170  0x019B  0       Suspend
 2   WKCN2.COM    0   512     0x41A0  0x41A0  0x017D  0       Suspend
 3   WKCN3.COM    0   512     0x41D0  0x41D0  0x01C2  0       Suspend
 4   WKCN4.COM    0   512     0x4200  0x4200  0x01B0  0       Suspend
 5   WKCN1.COM    0   512     0x4230  0x4230  0x019B  0       Suspend
 6   WKCN2.COM    0   512     0x4260  0x4260  0x017D  0       Suspend
 7   WKCN3.COM    0   512     0x4290  0x4290  0x01C2  0       Suspend
 8   WKCN4.COM    0   512     0x42C0  0x42C0  0x01B0  0       Suspend
wkcn >  _
```

杀死进程 1 和进程 2

输入 mem 命令，进行内存管理链表查询

```
wkcn > top
 There are 7 Progresses :-)
 PID Name         PR  Size    SEG     CS      IP      Parent  State
 0   Mirai-Shell  0   0       0x0000  0xEA60  0x25B2  0       Running
 3   WKCN3.COM    0   512     0x41D0  0x41D0  0x01C2  0       Suspend
 4   WKCN4.COM    0   512     0x4200  0x4200  0x01B0  0       Suspend
 5   WKCN1.COM    0   512     0x4230  0x4230  0x019B  0       Suspend
 6   WKCN2.COM    0   512     0x4260  0x4260  0x017D  0       Suspend
 7   WKCN3.COM    0   512     0x4290  0x4290  0x01C2  0       Suspend
 8   WKCN4.COM    0   512     0x42C0  0x42C0  0x01B0  0       Suspend
wkcn > mem
[16752, 16848) [17153, 33136)
Memory  : 250 / 256 Kbytes
MaxBlock: 249 Kbytes
wkcn > _
```

可以看见杀死进程后，内存链表不连续。

输入命令 box，（重复两次），然后按 Esc 键返回 Shell，输入 top

```
wkcn > top
 There are 9 Progresses :-)
 PID Name         PR  Size    SEG     CS      IP      Parent  State
 0   Mirai-Shell  0   0       0x0000  0xEA60  0x25B2  0       Running
 1   BOX.COM      0   327     0x4170  0x4170  0x023C  0       Suspend
 2   BOX.COM      0   327     0x4195  0x4195  0x023C  0       Suspend
 3   WKCN3.COM    0   512     0x41D0  0x41D0  0x01C2  0       Suspend
 4   WKCN4.COM    0   512     0x4200  0x4200  0x01B0  0       Suspend
 5   WKCN1.COM    0   512     0x4230  0x4230  0x019B  0       Suspend
 6   WKCN2.COM    0   512     0x4260  0x4260  0x017D  0       Suspend
 7   WKCN3.COM    0   512     0x4290  0x4290  0x01C2  0       Suspend
 8   WKCN4.COM    0   512     0x42C0  0x42C0  0x01B0  0       Suspend
wkcn > _
```

可见两个进程 BOX.COM 的 SEG 值分别为 0x4170, 0x4195.

即放在了以前两个被杀死的进程所占的内存资源中。

用户程序申请内存：

命令：

用户程序 可申请的最大内存

```
 wkcn > testnew
 Lack of Memory!
 wkcn > testnew 4
 9888109888830
 Lack of Memory!wkcn >
 wkcn > testnew 400
 9888109888830
 1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,
 wkcn > _
```

可以看见给予不同最大申请内存值的结果

注意到，testnew 申请了两个 uint16_t(每个 2 字节)，并释放。

然后申请一个 int 数组（共 400 字节）。当只给予 400 字节申请内存空间时，程序能够有效运行。说明内存管理机制有效！

六、小结

这次实验，我实现了信号量与内存管理。之所以实现内存管理，

是因为我对内存管理机制很感兴趣，之前也没学过连续区间删除合并的方法，为此想在自己的操作系统上尝试一下，也正好可以根据内存管理功能进行应用。我查看了关于内存管理的资料，了解到了很多种算法，权衡实现难度和功能效果，最终选择了比较简单的最佳适应法。我首先使用 C++编写了算法，在 Linux 上测试，基本无误后，再将其移植到我的操作系统上（该算法还有点瑕疵，暂时没有解决）。我花了两天多的时间实现，并且因为好奇 C++的 new 和 delete，自己也实现了一份。无奈 16 位下的内存寻址范围有限，只能把 C++申请的动态内存放在与进程相同的段内（因为指针是 16 位的）。如果使用保护模式的内存平坦模式，会方便很多。

对于信号量的实现，我觉得最大的问题是 G++加 NASM 的方式太坑了，在实现系统调用时花了很多精力。我觉得现在理解和技术都不是问题，我在 32 位 G++和 16 位 NASM 如何完美结合的方面花了大量时间，期间还想放弃，打算实现保护模式，免得出现那么多麻烦。但最终灵感+Bochs 反汇编，让我深刻的理解到 C++的编译器是如何生成指令，如何进行压栈的。（我以前写过脚本语言解释器，知道一点编译原理的知识）。

操作系统实验可能是我有史以来提交得最频繁的工程了。有时候，因为兴趣而做一件事情，我觉得很棒。