

实验八 进程同步机制 实验报告

数据科学与计算机学院 计算机科学与技术 2016 级
王凯祺 16337233

2018 年 5 月 16 日

1 实验目的

- 用于互斥和同步的计数信号量机制
- 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 实验要求

- 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 init_val) 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

3 实验步骤

3.1 解决历史遗留问题

老师提供的银行存款程序是长这样的：

```
1 include "process.h"
2 include "sync.h"
3 int bankbalance=1000; /*银行帐户余额1000元*/
4 void main() {
5     int pid, sem_id;
6     int t, totalsave=0, totaldraw=0;
7     sem_id=GetSem(1);
8     pid=fork();
9     if (pid==-1) {printf("error in fork!");exit(-1);}
10    if (pid) {
11        while (1) {
12            p(sem_id);
13            t=bankbalance; /*父进程反复存钱，每次10元*/
14            delay(3);
15            t=t+10;
```

```

16         delay(2)
17         bankbalance=t;
18         totalsave+=t;
19         printf( "bankbalance=%d,%totalsave=%d" , bankbalance,totalsave );
20         v(sem_id);
21     }
22     exit(0);
23 }
24 else {
25     /*子进程反复取钱，每次20元*/
26     exit(0);
27 }
28 }

```

我估算了一下这个程序编译后的大小，发现可能远大于 512 字节。由于我在实验三的时候过于自信地认为用户程序应该不会超过 512 字节，所以在内存里为每个用户程序都只预留了 512 字节的空间。在上个实验中，fork 测试程序已经差不多占满了，因此在这个实验中必须得为用户程序扩容。那么，我们需要重新布局软盘空间、重新分配内存空间、修改复制函数。

在实验七中我实现的 fork 是复制进程，而不是复制线程。具体为：复制 CS、DS、ES、SS 中的数据复制一份。现在由于要使用共享变量，必须使用线程而非进程。我只需要不再复制 CS、DS、ES，只复制 SS 中的数据。

3.2 Sleep 函数

老师的程序里使用了 Sleep 函数，是为了显著增加 RC 问题发生的概率。如果不加 Sleep 函数，此 RC 问题可能要花很长时间才能测试出来。

我使用系统调用来实现 Sleep 函数。用户程序调用 Sleep 函数时，启动中断服务程序。中断服务程序先做 save 操作，然后将当前进程阻塞，并记录该进程还需等待的秒数。每当时钟中断发生时，等待秒数 -1。当秒数减至 0 时，将当前进程恢复为就绪。

Sleep 中断核心代码展示：

```

1 void sleepint() {
2     int sleeptime;
3     save();
4     sleeptime = new_pcb.ax;
5     if (sleeptime > 0) {
6         PCBlist[now_process].status = BLOCK;
7         sleepqueue[sleepqueue_top].val = now_process;
8         sleepqueue[sleepqueue_top].sleeptime = sleeptime;
9         ++ sleepqueue_top;
10    }
11    exchange();
12 }

```

时钟中断核心代码展示：

```

1 void timeint() {
2     int i, j;
3     save();
4     for (i = 0; i < sleepqueue_top; ++i) {
5         sleepqueue[i].sleeptime -= 1;
6         if (sleepqueue[i].sleeptime <= 0) {
7             PCBlist[sleepqueue[i].val].status = READY;

```

```

8         for (j = i; j + 1 < sleepqueue_top; ++j) {
9             sleepqueue[j].val = sleepqueue[j + 1].val;
10            sleepqueue[j].sleeptime = sleepqueue[j + 1].sleeptime;
11        }
12        -- i;
13        -- sleepqueue_top;
14    }
15 }
16 exchange();
17 }

```

3.3 测试 RC 问题

我将老师的代码有关信号量的函数全部删去，测试一下是否真的会产生 RC 问题。
测试代码如下：

```

1  #include "stdlib.h"
2  #include "stdio.h"
3  #include "sync.h"
4
5  int bankbalance = 1000;
6
7  void syncmain() {
8      int pid, sem_id;
9      int t, i;
10     /* sem_id = getSem(1); */
11     if (sem_id >= 0 && sem_id < 64) {
12         puts_no_new_line("Applying_signal:_sem_id=_");
13         printint(sem_id);
14     } else {
15         puts("error_while_applying_signal");
16     }
17     pid = fork();
18     if (pid == -1) {
19         puts("error_in_fork!");
20         exit(-1);
21     }
22     if (pid) {
23         puts("father_process:_fork_OK");
24         sleep(2);
25         puts("father_process:_sleep_OK");
26         while (1) {
27             /* P(sem_id); */
28             t = bankbalance;
29             sleep(3);
30             t = t + 10;
31             sleep(2);
32             bankbalance = t;
33             puts_no_new_line("+10,_bankbalance=_");
34             printint(bankbalance);
35             /* V(sem_id); */
36         }
37         exit(0);

```

```

38     } else {
39         puts("child_process:_fork_OK");
40         sleep(2);
41         puts("child_process:_sleep_OK");
42         while (1) {
43             /*      P(sem_id);    */
44             t = bankbalance;
45             sleep(3);
46             t = t - 20;
47             sleep(2);
48             if (t >= 0) {
49                 bankbalance = t;
50                 puts_no_new_line("-20,_bankbalance=_");
51                 printint(bankbalance);
52             } else {
53                 puts("money_not_enough!");
54             }
55             /*      V(sem_id);    */
56         }
57         exit(0);
58     }
59 }

```

运行结果如下:

RC 问题在第一次取款的时候就出现了! 如果只有这一张截图, 并不能证明就是 RC 问题的锅。RC 问题的根源是两个进程共享变量, 如果这个变量是私有的, 那么上面这张截图是正常的。

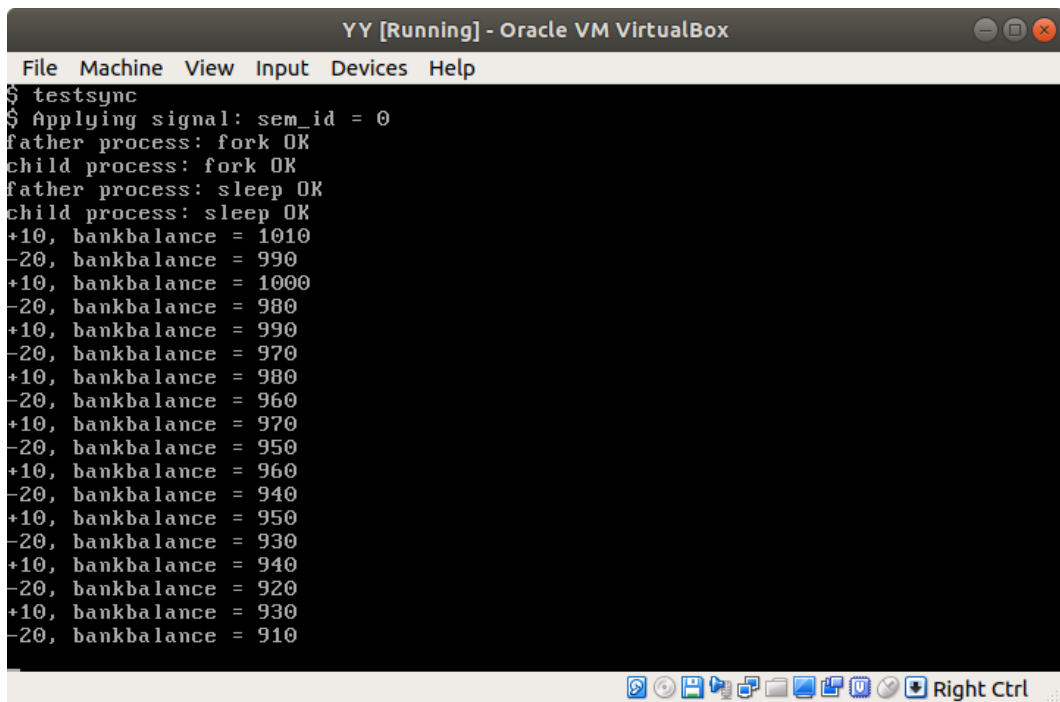

```

9         for (i = 0; i < 64; ++i)
10             if (signalav[i] == 0) {
11                 signalav[i] = 1;
12                 signal[i] = d;
13                 ok = i;
14                 break;
15             }
16     }
17     new_pcb.ax = ok;
18 } else
19 if (p == 0x1) {
20     if (d >= 0 && d < 64 && signalav[d]) {
21         -- signal[d];
22         if (signal[d] < 0) {
23             signalqueue[d][ signalqueue_top[d]++ ] = now_process;
24             PCBlist[now_process].status = BLOCK;
25         }
26     }
27 } else
28 if (p == 0x2) {
29     if (d >= 0 && d < 64 && signalav[d]) {
30         ++ signal[d];
31         if (signal[d] <= 0) {
32             PCBlist[ signalqueue[d][0] ].status = READY;
33             for (i = 0; i + 1 < signalqueue_top[d]; ++i)
34                 signalqueue[d][i] = signalqueue[d][i + 1];
35             -- signalqueue_top[d];
36         }
37     }
38 } else
39 if (p == 0x3) {
40     if (d >= 0 && d < 64 && signalav[d]) {
41         signalav[d] = 0;
42     }
43 }
44 exchange();
45 }

```

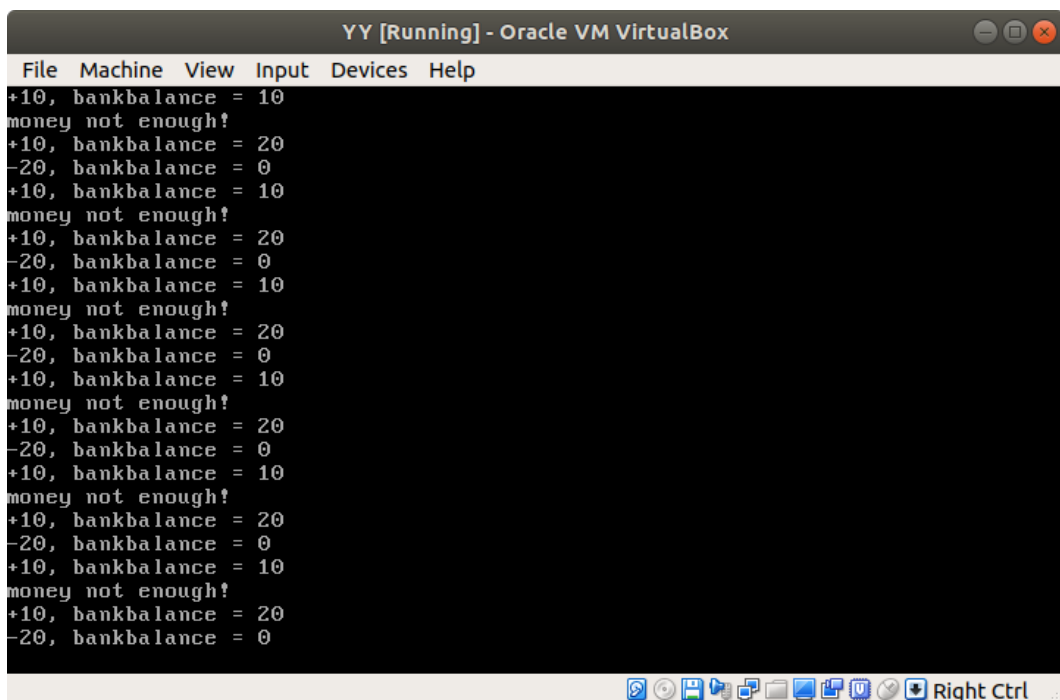
3.6 再次测试 RC 问题

做好信号量的工作后，将之前的测试代码加上信号量进行测试。



```
YY [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
$ testsync
$ Applying signal: sem_id = 0
father process: fork OK
child process: fork OK
father process: sleep OK
child process: sleep OK
+10, bankbalance = 1010
-20, bankbalance = 990
+10, bankbalance = 1000
-20, bankbalance = 980
+10, bankbalance = 990
-20, bankbalance = 970
+10, bankbalance = 980
-20, bankbalance = 960
+10, bankbalance = 970
-20, bankbalance = 950
+10, bankbalance = 960
-20, bankbalance = 940
+10, bankbalance = 950
-20, bankbalance = 930
+10, bankbalance = 940
-20, bankbalance = 920
+10, bankbalance = 930
-20, bankbalance = 910
```

在上图中我们可以看到 RC 问题已经得到解决。账平了！



```
YY [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
+10, bankbalance = 10
money not enough!
+10, bankbalance = 20
-20, bankbalance = 0
+10, bankbalance = 10
money not enough!
+10, bankbalance = 20
-20, bankbalance = 0
+10, bankbalance = 10
money not enough!
+10, bankbalance = 20
-20, bankbalance = 0
+10, bankbalance = 10
money not enough!
+10, bankbalance = 20
-20, bankbalance = 0
+10, bankbalance = 10
money not enough!
+10, bankbalance = 20
-20, bankbalance = 0
+10, bankbalance = 10
money not enough!
+10, bankbalance = 20
-20, bankbalance = 0
```

再上一张图，是儿子取完钱没钱了，RC 问题也没出问题。

4 实验总结

单纯信号量这个实验来说，在充分理解信号量原理的基础上实现，还是挺简单的。我花了 1 个小时写好调好 Sleep 函数，再花 1 个小时写好调好信号量。最花时间的还是解决历史遗留问题。特别是之前对未来的预判不足，导致资源不足，需要花大量的精力来扩展资源。同样的事情也发生在 Youtube，Youtube 没有预料到有视频的点赞数会超过 32 位整型的范围，故点赞数量是用

32 位整型存储的。结果那一夜，小苹果的视频点赞量爆 int 了……更新整个数据库是个非常庞大的工程。如果在之前有预见，做好扩展的准备，就不会这么麻烦。

同样的事情也应用在 FAT32 文件系统上，也是由于使用了 32 位整型，单文件大小最大为 4GB。随着科技的发展，4GB 的单文件大小明显不够用，那么又要写一套新的文件系统。我觉得程序的可扩展性是非常重要的，无论在何时，都不要想着用户资源“应该”不会超过多少 GB，而要及时保留扩展的余地（当然，我现在从原来假定用户程序不超过 512 字节改成假定用户程序不超过 4KB，也没有保留未来扩展的余地，要是程序超过 4KB，我又要去改内核了）。