

实现信号量机制与应用

实现信号量机制与应用

个人信息

实验题目

实验目的

实验要求

实验内容

实验方案

所用工具

虚拟机配置

代码概述

内核

进程模块

具体实现

fork的更新

wait的更新

exit的实现

信号量结构semaphore

GetSema的实现

FreeSema的实现

P操作实行

V操作实现

父子进程存取钱(无互斥)

父子进程存取钱(有互斥)

子进程送苹果祝福

读者写者问题(公平策略)

实验过程

生成com文件

写盘并且运行

运行虚拟机

实验总结

参考资料

个人信息

- 院系：数据科学与计算机学院
- 年纪：2018
- 姓名：王天龙
- 学号：18340168

实验题目

实现信号量机制与应用

实验目的

1. 理解基于信号量的进程/线程同步方法
2. 掌握内核实现信号量的方法
3. 掌握信号量的应用方法
4. 实现C库封装信号量相关系统调用

实验要求

1. 学习信号量机制原理
2. 扩展内核，设计实现一种计数信号量。提供信号量申请、初始化和p操作与v操作等功能的系统调用
3. 修改扩展C库，封装信号量相关的系统调用操作
4. 设计一个多线程同步应用的用户程序，展示你的多线程模型的应用效果
5. 编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

实验内容

1. 在没有互斥机制时，实际运行父子存款取款问题会产生竞态，设法截屏验证此事
2. 修改内核代码，增加信号量结构类型并定义一组信号量，组成信号量数组等数据结构。提供信号量申请初始化、释放和p操作与v操作等功能的系统调用
3. 修改扩展C库，封装信号量相关的系统调用操作
4. 设计一个多线程同步应用的用户程序，展示你的多线程模型的应用效果

应用1：用你的C库，编程解决多线程问题。进程创建2个线程，分别代表父亲和儿子利用一个银行账户存取资金。你先运行没有互斥同步代码，捕捉账户余额产生错误的运行过程，设法截屏验证此事。然后，在用信号量解决互斥同步，保证账户操作结果正确

应用2：用你的C库，编程解决多线程问题：父线程f创建二个子线程s和d，大儿子线程s反复向父线程f祝福，小儿进程d反复向父进程送水果(每次一个苹果或其他水果)，当二个子线程分别将一个祝福写到共享数组a和一个水果放进果盘(变量)后，父线程才去享受：从数组a收取出一个祝福和吃一个水果，如此反复进行若干次

应用3：读者-写者问题，公平方案


实验方案

所用工具

实验平台是windows10系统，使用的软件有：虚拟机软件vmware workstation pro、汇编语言编译器tasm、c语言编译器tcc、链接器tlink、可视化编译十六进制文件内容工具winhex、代码编辑器visual studio 2019

虚拟机配置

虚拟机为1cpu，内存为4MB，使用一个1.44MB大小的软盘，选择从软盘启动

设备	摘要
 内存	4 MB
 处理器	1
 CD/DVD (IDE)	自动检测
 软盘	正在使用文件 D:\computer_op...
 网络适配器	NAT
 声卡	自动检测
 显示器	自动检测

代码概述

内核

文件名	函数名/数据结构	功能
kernel.asm	_start	内核入口
kfun.asm	_clrIF	把IF标志位设为0，关中断
	_setIF	把IF标志位设为1，开中断
	_LoadPSP	把PSP加载到程序所在段的前0x100个字节
	_LoadINT20H	设置20h号中断，功能是返回操作系统
	_LoadINT22H	设置22h号中断，功能是打印字符串INT22H
	_putc	利用BIOS调用，显示一个字符
	_readkeypress	利用BIOS调用，从键盘读一个按键
	_cleanscreen	清屏
	_loadblock	从软盘加载扇区到指定内存
	_LoadINT8H	设置8h号中断，功能是风火轮以及时间片轮转
	_save	把当前进程的寄存器保存到PCB块，保护现场
	_restart	利用PCB块恢复线程
	_LoadINT21H	设置21h号中断，提供系统调用
ckfun.c	struct cpuState	保存所有寄存器的值
	struct PCB	进程控制块
	struct semaphore	信号量结构体
	semaphore semList[NRsem]	信号量队列
	PCB *currentPCB	指向当前正在运行的进程控制块
	PCB *kernelPCB	指向内核的进程控制块(把内核进程也视为一个普通进程)
	PCB pcbList[MaxProcessNum]	进程控制块队列
	char stackList[MaxProcessNum]	所有进程的栈段
	void strcpy(char* dest,char* source)	复制字符串
	void initPCBList()	初始化进程控制块
	void switchProcess()	切换进程控制块
	int creatProcess(int cs,int ip,int blockid,int numOfblock)	创建一个进程

文件名	函数名/数据结构	功能
	void endProcess()	系统调用，结束一个进程
	void printf(char* s)	打印一个字符串
	void scanf(char* s)	输入一个字符串
	void do_fork()	系统调用，从当前进程派生出一个子进程
	void do_exit()	系统调用，子进程退出
	void do_wait()	系统调用，父进程阻塞，等待子进程退出
	void initSemList()	初始化信号量队列
	void do_GetSema()	获取信号量
	void do_FreeSema()	释放信号量
	void do_P()	P操作
	void do_V()	V操作
	void kernelmain()	内核主函数

进程模块

文件名	函数	功能
process.asm	_fork	使用系统调用，从当前进程派生出子进程
	_exit	使用系统调用，结束子进程，同时唤醒父进程
	_wait	使用系统调用，使父进程阻塞，等待子进程结束时唤醒
	_GetSema	使用系统调用，获取一个信号量
	_FreeSema	使用系统调用，释放一个信号量
	_P	P操作
	_V	V操作

具体实现

这次实验进一步完善了五状态进程模型，同时对之前的fork、wait、exit函数进一步完善，增加了信号量以及对信号量的操作。

fork的更新

fork的更新在于，可以同时fork出多个线程，而且返回值也发生改变，-1为派生失败，0为父进程，1、2、3.....代表子进程的编号。

```

;process.asm
_fork proc near;一个参数，子进程的数量
    push bp
    mov bp,sp
    push bx
    mov bx,[bp+4];把参数放在bx寄存器里，do_fork时可以通过获取PCB块的bx的值得到
    mov ah,3h
    int 21h
    pop bx
    pop bp
    ret
_fork endp

```

内核中，3h号系统调用实现如下

```

;kfun.asm
SYCALL_3H proc near
    call _save;保护现场，这样在拷贝PCB块时，父子进程返回点相同

    call _do_fork;fork的具体过程用c语言实现

    jmp _restart
SYCALL_3H endp

```

do_fork函数

```

/*kfun.c*/
void do_fork()
{
    int i;
    int num = currentPCB->cpu.bx; /*获取要派生多少个子进程*/
    for (i = (currentPCB->pid + 1) % MaxProcessNum; i != currentPCB->pid; i++, i
    %= MaxProcessNum)
    {
        if (pcbList[i].state == 0) /*获取一个空的进程控制块*/
        {
            memcpy(&pcbList[currentPCB->pid], &pcbList[i], sizeof(PCB));
            memcpy(stackList[currentPCB->pid], stackList[i], sizeofStack);

            pcbList[i].state = 2;
            pcbList[i].fid = currentPCB->pid;
            pcbList[i].pid = i;
            pcbList[i].cpu.sp += ((i - currentPCB->pid) * sizeofStack); /*父子进程
            所在的段相同，但是栈顶指针不同，两个之间相差n个sizeofStack*/

            pcbList[i].cpu.ax = num; /*给子进程一个编号*/

            numOfProcess++;
            num--;
            if (num == 0) /*派生了num个子进程，可以返回了*/
            {
                currentPCB->cpu.ax = 0; /*父进程*/
                return;
            }
        }
    }
}

```

```

        currentPCB->cpu.ax = -1; /*创建失败*/
    }

```

wait的更新

在实验七中，wait函数非常粗糙，很多情况没有考虑到。例如，如果有多个子进程，而每个子进程结束都会唤醒父进程一次，那么父进程需要多次使用系统使用系统调用使自己阻塞；还有就是如果父进程执行wait函数时，子进程已经返回，因此，系统调用不能直接把父进程设为阻塞，而是想要检查子进程是否还在运行或者就绪或者阻塞。

```

;process.asm
_wait proc near
@1:
    mov ah,5h
    int 21h
    cmp ax,1
    jnz @1;如果所有子进程都已经退出，则5h号系统调用的返回值ax==1，否则，会继续使用系统调用
    ret
_wait endp

```

内核中，5h号系统调用实现如下

```

;kfun.asm
SYCALL_5H proc near
    call _save
    call _do_wait;把父进程设为阻塞，并且调用了switchProcess，切换了进程
    jmp _restart;_restart恢复的是另一个进程
SYCALL_5H endp

```

do_wait函数

```

/*ckfun.c*/
void do_wait()
{
    /*currentPCB->state = 3;
    switchProcess();*/
    int i;
    for (i = 0; i < MaxProcessNum; i++)
    {
        if (pcbList[i].state != 0 && pcbList[i].fid == currentPCB->pid && i !=
currentPCB->pid)/*寻找这个父进程的所有子进程*/
        {
            currentPCB->state = 3;
            currentPCB->cpu.ax = 0;
            switchProcess();
            return;
        }
    }
    currentPCB->cpu.ax = 1; /*已经没有子进程了，则返回-1*/
}

```

exit的实现

实验七中的exit函数同样有一个漏洞，即如果父进程已经退出了，则exit函数不应该把它唤醒。更新如下

do_exit函数

```
void do_exit()
{
    if (pcbList[currentPCB->fid].state == 3)/*当父进程处于阻塞时，才需要唤醒*/
    {
        pcbList[currentPCB->fid].state = 2;
    }
    currentPCB->state = 0;
    numOfProcess--;
}
```

信号量结构semaphore

内核中，定义了信号量结构体，以及信号量队列

```
#define NRsem 10/*信号量个数*/

typedef struct/*信号量*/
{
    int count;/*信号量的值*/
    PCB *next;/*阻塞队列*/
    char used;/*为0时说明该信号量没有被占用，为1时说明被占用*/
}semaphore;

semaphore semList[NRsem];/*信号量队列*/
```

GetSema的实现

使用这个函数获取一个信号量，有一个参数，即信号量的初始值，返回值为-1时，说明获取失败，否则是信号量在队列里的编号，具体实现如下

```
;process.asm
_GetSema proc near
    push bp
    mov bp,sp
    push bx
    mov bx,[bp+4];获取参数，即信号量的初始值
    mov ah,6h;使用6h号系统调用，获取信号量
    int 21h
    pop bx
    pop bp
    ret
_GetSema endp
```

内核中，6h号系统调用实现

```
;kfun.asm
SYCALL_6H proc near
    call _save
    call _do_GetSema;c语言实现
    jmp _restart
SYCALL_6H endp
```

do_GetSema实现

```

/*ckfun.c*/
void do_GetSema()/*参数s在currentPCB->bx里*/
{
    int i = 0;
    for (; i < NRsem; i++)
    {
        if (semList[i].used == 0)/*找到一个没被占用的信号量*/
        {
            semList[i].used = 1;
            semList[i].count = currentPCB->cpu.bx;/*获取初始值*/
            currentPCB->cpu.ax = i;/*返回该信号量的编号*/
            return;
        }
    }
    currentPCB->cpu.ax = -1;/*获取信号量失败*/
}

```

FreeSema的实现

这个用于释放一个信号量，一般有父进程调用，而且父进程调用前，已经确保子进程都已经结束，即调用了wait函数

```

;process.asm
_FreeSema proc near
    push bp
    mov bp,sp
    push bx
    mov bx,[bp+4];获取参数，说明要释放哪一个信号量，并放在bx寄存器中
    mov ah,7h;调用7h号系统调用
    int 21h
    pop bx
    pop bp
    ret
_FreeSema endp

```

内核中，7h号系统调用实现

```

;kfun.asm
SYCALL_7H proc near
    call _save
    call _do_FreeSema;c语言实现
    jmp _restart
SYCALL_7H endp

```

do_FreeSema函数

```

void do_FreeSema()
{
    /*bx寄存器里就是要释放的信号量*/
    semList[currentPCB->cpu.bx].used = 0;
}

```

P操作实行

对一个信号量进行P操作，如果信号量的值小于0则阻塞当前进程


```

;process.asm
_P proc near
    push bp
    mov bp,sp
    push bx
    mov bx,[bp+4];获取参数，即要执行P操作的信号量，并且放到bx寄存器中
    mov ah,8h;使用8h号系统调用
    int 21h
    pop bx
    pop bp
    ret
_P endp

```

内核中，8h号中断的思想

```

;kfun.asm
SYCALL_8H proc near
    call _save
    call _do_P;c语言实现
    jmp _restart
SYCALL_8H endp

```

do_P函数

```

/*ckun.c*/
void do_P()
{
    semList[currentPCB->cpu.bx].count--; /*信号量的值减一*/
    if (semList[currentPCB->cpu.bx].count < 0)
    {
        currentPCB->state = 3; /*阻塞当前进程*/
        currentPCB->next = semList[currentPCB->cpu.bx].next; /*把当前进程加到阻塞队列里*/
        semList[currentPCB->cpu.bx].next = currentPCB;
        switchProcess(); /*切换进程*/
    }
}

```

V操作实现

对一个信号量进行V操作，如果信号量的值小于或等于0，则从阻塞队列里取出一个进程唤醒

```

;process.asm
_V proc near
    push bp
    mov bp,sp
    push bx
    mov bx,[bp+4];取得参数，即要执行V操作的信号量，并把它放到bx寄存器中
    mov ah,9h;使用9h号系统调用
    int 21h
    pop bx
    pop bp
    ret
_V endp

```

内核中，9h号系统调用的实现

```
;kfun.asm
SYCALL_9H proc near
    call _save
    call _do_v; c语言实现
    jmp _restart
SYCALL_9H endp
```

do_V函数

```
/*ckfun.c*/
void do_V()
{
    PCB *temp;
    semList[currentPCB->cpu.bx].count++; /*信号量的值加一*/
    if (semList[currentPCB->cpu.bx].count <= 0)
    {
        /*从阻塞队列中唤醒一个进程*/
        temp = semList[currentPCB->cpu.bx].next;
        semList[currentPCB->cpu.bx].next = semList[currentPCB->cpu.bx].next-
>next;
        temp->state = 2;
    }
}
```

父子进程存取钱(无互斥)

父亲和儿子利用一个引号账号存取资金。父亲存钱，每次存10元；儿子取钱，每次取20元；账号初始余额为1000元。无互斥保护

```
/*ptest1.c*/
/*父亲和儿子利用同一个银行账号存取资金，无互斥同步*/

extern int fork(int num);
extern void printf(char *s, ...);
extern void exit();
extern void wait();

void sleep(int time) /*使用sleep函数以观察出错*/
{
    int i, j;
    for (i = 0; i < time * 10000; i++)
    {
        for (j = 0; j < 10000; j++);
    }
}

int bankBalance = 1000; /*账号余额*/

int main()
{
    int pid;
    int i;
    int t, totalSave = 0, totalDraw = 0;
    pid = fork(1);
```

```

if (pid == -1)
{
    printf("error in fork\r\n");
    return;
}
if (pid == 0)/*父进程存钱，每次10元*/
{
    for (i = 0; i < 10; i++)
    {
        t = bankBalance;/*查询余额*/
        sleep(2);
        t += 10;/*存钱*/
        sleep(2);
        bankBalance = t;/*银行更新存款*/
        totalSave += 10;/*记录一共存了多少钱*/
        printf("bankBalance=%d,totalSave=%d\r\n", bankBalance, totalSave);
    }
    wait();/*等待子进程结束*/
}
else if (pid == 1)/*子进程取钱，每次20*/
{
    for (i = 0; i < 10; i++)
    {
        t = bankBalance;/*查询余额*/
        sleep(2);
        t -= 20;/*取钱*/
        sleep(2);
        bankBalance = t;/*银行更新存款*/
        totalDraw += 20;/*记录一共取了多少钱*/
        printf("bankBalance=%d,totalDraw=%d\r\n", bankBalance, totalDraw);
    }
    exit();/*子进程退出*/
}
}

```

父子进程存取钱(有互斥)

父亲和儿子利用一个引号账号存取资金。父亲存钱，每次存10元；儿子取钱，每次取20元；账号初始余额为1000元。有互斥保护，实现互斥保护是利用信号量，创造出一个临界区，只能由一个进程执行。

```

/*ptest2.c*/
/*父亲和儿子利用同一个银行账号存取资金，有互斥同步*/

extern int fork(int num);
extern void wait();
extern void exit();
extern int GetSema(int value);
extern void FreeSema(int s);
extern void P(int s);
extern void V(int s);
extern void printf(char *s, ...);

void sleep(int time)
{
    int i, j;

```

```

    for (i = 0; i < time * 10000; i++)
    {
        for (j = 0; j < 10000; j++);
    }
}

int bankBalance = 1000; /*账号余额*/
int main()
{
    int pid;
    int i;
    int s; /*信号量*/
    int t, totalSave = 0, totalDraw = 0;
    s = GetSema(1); /*获取信号量*/
    pid = fork(1); /*注意，要在fork之前就获取信号量，否则或获取多个信号量*/
    if (pid == -1)
    {
        printf("error in fork\r\n");
        return;
    }
    if (pid == 0) /*父进程存钱，每次10元*/
    {
        for (i = 0; i < 10; i++)
        {
            P(s); /*进入临界区*/
            t = bankBalance; /*查询余额*/
            sleep(2);
            t += 10; /*存钱*/
            sleep(2);
            bankBalance = t; /*银行系统更新余额*/
            totalSave += 10;
            printf("bankBalance=%d,totalSave=%d\r\n", bankBalance, totalSave);
            V(s); /*退出临界区*/
        }
        wait();
        FreeSema(s); /*父进程等待子进程结束后，再释放信号量*/
    }
    else if (pid == 1) /*子进程取钱，每次20*/
    {
        for (i = 0; i < 10; i++)
        {
            P(s); /*进入临界区*/
            t = bankBalance; /*查询余额*/
            sleep(2);
            t -= 20; /*取钱*/
            sleep(2);
            bankBalance = t; /*银行更新余额*/
            totalDraw += 20;
            printf("bankBalance=%d,totalDraw=%d\r\n", bankBalance, totalDraw);
            V(s); /*退出临界区*/
        }
        exit();
    }
}

```

子进程送苹果祝福

父线程f创建二个子线程s和d，大儿子线程s反复向父线程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)，当二个子线程分别将一个祝福写到共享数组a和一个水果放进果盘(变量)后，父线程才去享受：从数组a收取出一个祝福和吃一个水果，如此反复进行若干次

要实现上述过程，需要用到两个信号量，一个代表水果、一个代表祝福，当父进程获取了两个信号量后，查看祝福和水果，当两个都齐全时才享受

```
/*ptest3.c*/

extern int fork(int num);/*要改进fork，让它可以fork多个进程*/
extern void wait();
extern void exit();
extern int GetSema(int value);
extern void FreeSema(int s);
extern void P(int s);
extern void V(int s);
extern void printf(char *s, ...);

void strcpy(char *dst, char *src)/*复制字符串，即子进程送祝福过程*/
{
    int i;
    for (i = 0; src[i] != '\0'; i++)
    {
        dst[i] = src[i];
    }
    dst[i] = '\0';
}

char words[80];/*祝福，当首字节为0时，说明无祝福*/
char fruit;/*0 for empty, 1 for apple, 2 for banana*/

int main()
{
    int pid;
    int fs, ws; /*两个信号量，分别代表水果和祝福*/
    int i;
    words[0] = '\0'; /*刚开始时，没有祝福也没有水果*/
    fruit = 0;
    fs = GetSema(1); /*申请信号量*/
    ws = GetSema(1);
    pid = fork(2);
    if (pid == -1)
    {
        printf("error in fork\r\n");
        return;
    }
    if (pid == 0) /*父亲*/
    {
        for (i = 0; i < 10; i++)
        {
            P(ws); /*获取两个信号量*/
            P(fs);

            if (words[0] == 0 || fruit == 0)
            {
                i--;
            }
        }
    }
}
```

```

else/*当水果和祝福都齐全时，才去享受*/
{
    if (fruit == 1)
    {
        printf("words: %s fruit: apple\r\n", words);
    }
    else
    {
        printf("words: %s fruit: banana\r\n", words);
    }
    words[0] = '\0';/*取走祝福和水果*/
    fruit = 0;
}

v(fs);/*释放两个信号量*/
v(ws);
}
wait();
FreeSema(fs);
FreeSema(ws);
}
else if (pid == 1)/*大儿子s*/
{
    for (i = 0; i < 10; i++)
    {
        P(ws);
        if (words[0] != '\0')/*如果祝福没有被取走，则下次再送*/
        {
            i--;
        }
        else/*送祝福*/
        {
            strcpy(words, "Father will live one year after another for
ever!");
        }
        v(ws);
    }
    exit();
}
else if (pid == 2)/*小儿子d*/
{
    for (i = 0; i < 10; i++)
    {
        P(fs);
        if (fruit != 0)/*如果水果没有被取走则下次再送*/
        {
            i--;
        }
        else
        {
            fruit = i % 2 + 1;
        }
        v(fs);
    }
    exit();
}
}
}

```

读者写者问题(公平策略)

使用信号量解决读者-写者问题，采用公平策略。

采用公平策略，则读者和写者有相同的优先权，读者和写者按照先后顺序对数据进行读和写。为此，需要2个信号量，一个用于互斥写者和读者，一个用于互斥写者和写者，同时，由于要记录正在读的读者的数量，所以还需要一个信号量用于读者之间能正确更新该数值。具体如下

```
/*ptest4.c*/
/*读者写者问题，公平方案*/

/*3个读者，2个写者*/
#define ReaderNum 3
#define WriterNum 2

extern int fork(int num);
extern void wait();
extern void exit();
extern int GetSema(int value);
extern void FreeSema(int s);
extern void P(int s);
extern void V(int s);
extern void printf(char *s, ...);

int reader_counter = 0; /*记录正在读的读者*/

int main()
{
    int wmutex, rmutex, mutex; /*互斥写者，互斥读者(确保正确更新reader_counter)，互斥读
者和写者*/
    int pid;
    int i;
    wmutex = GetSema(1);
    rmutex = GetSema(1);
    mutex = GetSema(1);
    pid = fork(ReaderNum + WriterNum); /*0 for father, 1-ReaderNum for reader,
other for writer*/
    if (pid == -1)
    {
        printf("error in fork\r\n");
        return;
    }
    if (pid == 0) /*father*/
    {
        wait();
        FreeSema(wmutex);
        FreeSema(rmutex);
        FreeSema(mutex);
    }
    else if (1 <= pid && pid <= ReaderNum) /*reader*/
    {
        for (i = 0; i < 2; i++)
        {
            /*读者进入*/
            P(rmutex); /*保证reader_counter正确更新*/
            if (reader_counter == 0) /*这是第一个到来的读者*/
            {
                P(mutex); /*当写者写完时，读者才可以读*/
            }
        }
    }
}
```

```

    }
    reader_counter++;/*读者数量增加*/
    V(rmutex);
    /*此时其它读者可以进入*/
    /*读操作*/
    printf("reader %d is reading, num of readers %d\r\n", pid - 1,
reader_counter);

    /*读者退出*/
    P(rmutex);
    reader_counter--;
    if (reader_counter == 0)/*这是最后一个读者*/
    {
        V(mutex);/*当所有进入的读者都读完时，写者才可以写*/
    }
    V(rmutex);
}
exit();
}
else/*writer*/
{
    for (i = 0; i < 3; i++)
    {
        P(wmutex);/*只有一个写者可以进入*/
        P(mutex);/*当队列前面读者读完之后，才可以写*/
        /*写操作*/
        printf("writer %d is writing\r\n", pid - ReaderNum - 1);
        V(mutex);
        V(wmutex);
    }
    exit();
}
}
}

```

实验过程

生成com文件

在DoxBox里使用命令

```

tasm kernel.asm
tasm kfun.asm
tcc -c ckfun.c
tlink /3 /t kernel.obj kfun.obj ckfun.obj, kernel.com

```

即可生成内核程序的com执行体。用户程序也类似。

写盘并且运行

写盘式，引导程序在首扇区，内核在2、3、4、5、6、7扇区，ptest1在8、9、10号扇区；ptest2在11、12、13号扇区；ptest3在14、15、16号扇区；ptest4在17、18、19号扇区。可以用指令查看所有程序。

运行虚拟机

运行ptest1，父子进程存取钱(无互斥)，输入1


```

M for message, l to list, c to clean screen, 1-4 to select program: 1
bankBalance=1010,totalSave=10
bankBalance=980,totalDraw=20
bankBalance=1020,totalSave=20
bankBalance=960,totalDraw=40
bankBalance=1030,totalSave=30
bankBalance=940,totalDraw=60
bankBalance=1040,totalSave=40
bankBalance=920,totalDraw=80
bankBalance=1050,totalSave=50
bankBalance=900,totalDraw=100
bankBalance=1060,totalSave=60
bankBalance=880,totalDraw=120
bankBalance=1070,totalSave=70
bankBalance=860,totalDraw=140
bankBalance=1080,totalSave=80
bankBalance=840,totalDraw=160
bankBalance=1090,totalSave=90
bankBalance=820,totalDraw=180
bankBalance=1100,totalSave=100
bankBalance=800,totalDraw=200
M for message, l to list, c to clean screen, 1-4 to select program: _

```

结果错误，两个进程的余额完全不同

运行ptest2，父子进程存取钱(有互斥)，输入2

```

M for message, l to list, c to clean screen, 1-4 to select program: 2
bankBalance=1010,totalSave=10
bankBalance=990,totalDraw=20
bankBalance=1000,totalSave=20
bankBalance=980,totalDraw=40
bankBalance=990,totalSave=30
bankBalance=970,totalDraw=60
bankBalance=980,totalSave=40
bankBalance=960,totalDraw=80
bankBalance=970,totalSave=50
bankBalance=950,totalDraw=100
bankBalance=960,totalSave=60
bankBalance=940,totalDraw=120
bankBalance=950,totalSave=70
bankBalance=930,totalDraw=140
bankBalance=940,totalSave=80
bankBalance=920,totalDraw=160
bankBalance=930,totalSave=90
bankBalance=910,totalDraw=180
bankBalance=920,totalSave=100
bankBalance=900,totalDraw=200
M for message, l to list, c to clean screen, 1-4 to select program: _

```

运行ptest3，子进程送苹果祝福，输入3

```

M for message, l to list, c to clean screen, 1-4 to select program: 3
words: Father will live one year after anther for ever! fruit: apple
words: Father will live one year after anther for ever! fruit: banana
words: Father will live one year after anther for ever! fruit: apple
words: Father will live one year after anther for ever! fruit: banana
words: Father will live one year after anther for ever! fruit: apple
words: Father will live one year after anther for ever! fruit: banana
words: Father will live one year after anther for ever! fruit: apple
words: Father will live one year after anther for ever! fruit: banana
words: Father will live one year after anther for ever! fruit: apple
words: Father will live one year after anther for ever! fruit: banana
M for message, l to list, c to clean screen, 1-4 to select program:

```

运行ptest4，读者写者问题(公平策略)，输入4

```

M for message, l to list, c to clean screen, 1-4 to select program: 4
writer 1 is writing
writer 1 is writing
writer 1 is writing
writer 0 is writing
writer 0 is writing
writer 0 is writing
reader 2 is reading, num of readers 1
reader 2 is reading, num of readers 1
reader 1 is reading, num of readers 1
reader 1 is reading, num of readers 1
reader 0 is reading, num of readers 1
reader 0 is reading, num of readers 1
M for message, l to list, c to clean screen, 1-4 to select program:

```

结果正常

实验总结

经过这次实验，我对多进程模型有了进一步理解，尤其是信号量机制。同时，我对读者-写者问题的认识更加清晰，掌握了三种策略问题。

参考资料

<https://blog.csdn.net/yanfeivip8/article/details/12527047>

https://blog.csdn.net/qg_35235032/article/details/106652964?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param