实验2进程PV操作

**一、实验目的**

1. 掌握信号通信机制，实现进程之间通过信号进行通信；

2. 掌握共享内存、信号量通信实现方法。

**二、实验工具与设备**

装有Linux系统的计算机。

**三、实验内容**

**首先了解一下，信号量机概念是由荷兰科学家Dijkstr引入，值得一提的是，它提出的Dijksrtr算法解决了最短路径问题。**

信号量又称为**信号灯**，它是用来协调不同进程间的数据对象的，而最主要的应用是共享内存方式的进程间通信。本质上，信号量是一个计数器，**它用来记录对某个资源（如共享内存）的存取状况**，信号量是一个特殊的变量，并且只有两个操作可以改变其值：等待(wait)与信号(signal)。

因为在Linux与UNIX编程中，"wait"与"signal"已经具有特殊的意义了（所以原始概念：   
     用于等待(wait)的P(信号量变量) ；   
     用于信号(signal)的V(信号量变量) ；   
这两字母来自等待(passeren：通过，如同临界区前的检测点)与信号(vrjgeven：指定或释放，如同释放临界区的控制权)的荷兰语。

P操作 负责把当前进程由运行状态转换为阻塞状态，直到另外一个进程唤醒它。

操作为：申请一个空闲资源（把信号量减1），若成功，则退出；若失败，则该进程被阻塞；

V操作 负责把一个被阻塞的进程唤醒，它有一个参数表，存放着等待被唤醒的进程信息。

操作为：释放一个被占用的资源（把信号量加1），如果发现有被阻塞的进程，则选择一个唤醒之。

**（一）系统调用函数semget()**

函数原型：int semget(key\_t key,int nsems,int semflg);

功能描述： 创建一个新的信号量集，或者存取一个已经存在的信号量集。

当调用semget创建一个信号量时，他的相应的semid\_ds结构被初始化。ipc\_perm中各个量被设置为相应   
值：   
        sem\_nsems被设置为nsems所示的值；       
        sem\_otime被设置为0；    
        sem\_ctime被设置为当前时间

参数介绍：   
         key：所创建或打开信号量集的键值,键值是IPC\_PRIVATE，该值通常为0，创建一个仅能被进程进程给我的信号量, 键值不是IPC\_PRIVATE，我们可以指定键值，例如1234；也可以一个ftok()函数来取得一个唯一的键值。   
         nsems：创建的信号量集中的信号量的个数，该参数只在创建信号量集时有效。   
         semflg：调用函数的操作类型，也可用于设置信号量集的访问权限，两者通过or表示：

                有IPC\_CREAT，IPC\_EXCL两种：

IPC\_CREAT如果信号量不存在，则创建一个信号量，否则获取。

IPC\_EXCL只有信号量不存在的时候，新的信号量才建立，否则就产生错误。

返回值说明：   
如果成功，则返回信号量集的IPC标识符，其作用与信息队列识符一样。   
如果失败，则返回-1，errno被设定成以下的某个值   
EACCES：没有访问该信号量集的权限   
EEXIST：信号量集已经存在，无法创建   
EINVAL：参数nsems的值小于0或者大于该信号量集的限制；或者是该key关联的信号量集已存在，并且nsems   
大于该信号量集的信号量数   
ENOENT：信号量集不存在，同时没有使用IPC\_CREAT   
ENOMEM ：没有足够的内存创建新的信号量集   
ENOSPC：超出系统限制

图解：

|  |
| --- |
|  |
| 每个信号量都有一些相关值：        semval 信号量的值，一般是一个正整数，它只能通过信号量系统调用semctl函数设置，程序无法直接对它进行修改。        sempid 最后一个对信号量进行操作的进程的pid.        semcnt 等待信号量的值大于其当前值的进程数。        semzcnt 等待信号量的值归零的进程数。 |  |

**（二）信号量的控制 semctl()**

原型：int semctl(int semid,int semnum,int cmd,union semun ctl\_arg);    
参数介绍： semid为信号量集引用标志符，即semget 的返回值。    
               semnum第二个参数是信号量数目；

               cmd表示调用该函数执行的操作，其取值和对应操作如下：

|  |
| --- |
| **标准的IPC函数**  （注意在头文件<sys/sem.h>中包含semid\_ds结构的定义） |
| IPC\_STAT 把状态信息放入ctl\_arg.stat中  IPC\_SET 用ctl\_arg.stat中的值设置所有权/许可权  IPC\_RMID 从系统中删除信号量集合 |
| **单信号量操作**  （下面这些宏与sem\_num指定的信号量合semctl返回值相关） |
| GETVAL 返回信号量的值（也就是semval）  SETVAL 把信号量的值写入ctl\_arg.val中  GETPID 返回sempid值  GETNCNT 返回semncnt（参考上面内容）  GETZCNT 返回semzcnt（参考上面内容） |
| **全信号量操作** |
| GETALL 把所有信号量的semvals值写入ctl\_arg.array  SETALL 用ctl\_arg.array中的值设置所有信号量的semvals |

参数arg代表一个union的semun的实例。semun是在linux/sem.h中定义的：

union semun {   
int val; //执行SETVAL命令时使用   
struct semid\_ds \*buf; //在IPC\_STAT/IPC\_SET命令中使用   
unsigned short \*array; //使用GETALL/SETALL命令时使用的指针   
}

联合体中每个成员都有各自不同的类型，分别对应三种不同的semctl 功能，如果semval 是SETVAL.则使用的将是ctl\_arg.val.

。

功能：smctl函数依据command参数会返回不同的值。它的一个重要用途是为信号量赋初值，因为进程无法直接对信号量的值进行修改。

**（三）信号量操作semop函数**

在 Linux 下，PV 操作通过调用**semop**函数来实现，也只有它能对PV进行操作

调用原型：int semop(int semid,struct sembuf\*sops,unsign ednsops);   
返回值：0，如果成功。-1，如果失败：errno=E2BIG(nsops大于最大的ops数目)   
EACCESS(权限不够)   
EAGAIN(使用了IPC\_NOWAIT，但操作不能继续进行)   
EFAULT(sops指向的地址无效)   
EIDRM(信号量集已经删除)   
EINTR(当睡眠时接收到其他信号)   
EINVAL(信号量集不存在,或者semid无效)   
ENOMEM(使用了SEM\_UNDO,但无足够的内存创建所需的数据结构)   
ERANGE(信号量值超出范围)

参数介绍：

第一个参数semid 是信号量集合标识符，它可能是从前一次的semget调用中获得的。

第二个参数是一个sembuf结构的数组，每个 sembuf 结构体对应一个特定信号的操作，sembuf结构在，<sys/sem.h>中定义

struct sembuf{   
usign short sem\_num;/\*信号量索引\*/   
short sem\_op;/\*要执行的操作\*/   
short sem\_flg;/\*操作标志\*/   
}

sem\_num 存放集合中某一信号量的索引，如果集合中只包含一个元素，则sem\_num的值只能为0。

----------------------------------------------------------------------------------------------

Sem\_op取得值为一个有符号整数，该整数实际给定了semop函数将完成的功能。包括三种情况：

      如果sem\_op是负数，那么信号量将减去它的值，对应于p（）操作。这和信号量控制的资源有关。如果没有使用IPC\_NOWAIT，那么调用进程将进入睡眠状态，直到信号量控制的资源可以使用为止。

      如果sem\_op是正数，则信号量加上它的值。对应于v（）操作。这也就是进程释放信号量控制的资源。

      最后，如果sem\_op是0，那么调用进程将调用sleep()，直到信号量的值为0。这在一个进程等待完全空闲的资源时使用。

----------------------------------------------------------------------------------------------

sem\_flag是用来告诉系统当进程退出时自动还原操作，它维护着一个整型变量semadj（信号灯的计数器），可设置为 IPC\_NOWAIT 或 SEM\_UNDO 两种状态。只有将 sem\_flg 指定为 SEM\_UNDO 标志后，semadj （所指定信号量针对调用进程的调整值）才会更新，即减去减去sem\_num的值。 此外，如果此操作指定SEM\_UNDO，系统更新过程中会撤消此信号灯的计数（semadj）。此操作可以随时进行---它永远不会强制等待的过程。调用进程必须有改变信号量集的权限。

第三个参数是sembuf组成的数组中索引。参数sops指向由sembuf组成的数组，结构数组中的一员。

**实验代码：**

实验所需头文件:

pv.h:

//pv.h头文件

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef union \_semun

{

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

信号量赋初值以及获取信号量标识符函数：

initsem.c

#include "pv.h"

int initsem(key\_t semkey)

{

int status=0,semid; //信号量标识符semid

if ((semid=semget(semkey,1,SEMPERM|IPC\_CREAT|IPC\_EXCL))==-1)

{

if (errno==EEXIST) //EEXIST：信号量集已经存在，无法创建

semid=semget(semkey,1,0); //创建一个信号量

}

else

{

semun arg;

arg.val=1; //信号量的初值

status=semctl(semid,0,SETVAL,arg); //设置信号量集中的一个单独的信号量的值。

}

if (semid==-1||status==-1)

{

perror("initsem failed");

return(-1);

}

/\*all ok\*/

return(semid);

}

v操作

v.c:

#include "pv.h"

int v(int semid)

{

struct sembuf v\_buf;

v\_buf.sem\_num=0;

v\_buf.sem\_op=1; //信号量加1

v\_buf.sem\_flg=0;

if (semop(semid, &v\_buf, 1)==-1)

{

perror("v(semid)failed");

exit(1);

}

return(0);

}

p操作

p.c

#include "pv.h"

int p(int semid)

{

struct sembuf p\_buf;

p\_buf.sem\_num=0;

p\_buf.sem\_op=-1; //信号量减1，注意这一行的1前面有个负号

p\_buf.sem\_flg=0;

//p\_buf = {0,-1,SEM\_UNDO};

if (semop(semid, &p\_buf, 1)==-1)

{

perror("p(semid)failed");

exit(1);

}

return(0);

}

测试函数一（使用PV操作实现三个进程的互斥）

/testsem.c 主程序，使用PV操作实现三个进程的互斥

testsem.c:

#include "pv.h"

void handlesem(key\_t skey);

int main()

{

key\_t semkey=0x200;

int i;

for (i=0;i<3;i++)

{

if (fork()==0) //父进程负责产生3个子进程

handlesem(semkey); //子进程中才执行handlesem，做完后就exit。

}

}

void handlesem(key\_t skey)

{

int semid;

pid\_t pid=getpid();

if ((semid=initsem(skey))<0)

exit(1);

printf("进程 %d 在临界资源区之前 \n",pid);

p(semid); //进程进入临界资源区，信号量减少1

printf("进程 %d 在使用临界资源时，停止10s \n",pid);

/\*in real life do something interesting \*/

sleep(10);

printf("进程 %d 退出临界区后 \n",pid);

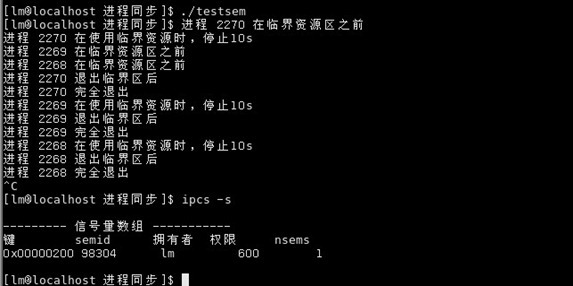
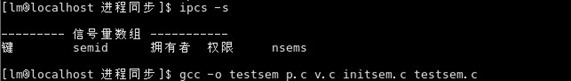
v(semid); //进程退出临界资源区，信号量加1

printf("进程 %d 完全退出\n",pid);

exit(0);

}

测试结果截图：



测试函数二（实现两个进程交替输出A和B，并在程序中查看信号量的值）

//ab.c 主程序，使用PV操作，两个进程交替输出A和B，实现临界区的互斥访问的基本模型

ab.c:

#include "pv.h"

i

nt main()

{

key\_t semkey\_A=0x200;

key\_t semkey\_B=0x220;

int semid\_A,semid\_B;

if ((semid\_A=initsem(semkey\_A,1))<0) exit(1);

if ((semid\_B=initsem(semkey\_B,0))<0) exit(1);

printf("A de %d,chu shi zhi %d\n",

semid\_A,semctl(semid\_A, 0, GETVAL));

printf("B de %d,chu shi zhi %d\n",

semid\_B,semctl(semid\_B, 0, GETVAL));

if (fork()!=0) //父进程先执行

{

int i;

for (i=0;i<10;i++)

{

p(semid\_A);

printf("A de %d\n",semctl(semid\_A, 0, GETVAL));

v(semid\_B);

}

}

else

{

int j;

for (j=0;j<10;j++)

{

p(semid\_B);

printf("Bde %d\n",semctl(semid\_A, 0, GETVAL));

v(semid\_A);

}

}

}

测试结果

