信号量（SEM）

一.概念

- **信号量的作用**

  信号量SEM全称Semaphore，中文也翻译为信号灯。

  作为system-V IPC通信的最后一种，信号量跟前面的MSG和SHM有极大的不同，SEM不是用来传输数据的，

  它是在多线程或多进程环境下使用的一种措施，用来协调各进程或线程在通信时对于共享资源的访问。

- **概念扫盲**

  Linux中用到的信号量有3种: ststem-V信号量、POSIX有名信号量和POSIX无名信号量。

  他们虽然有很多显著不同的地方，但是最基本的功能是一致的:

  用来表征一种资源的数量，当多个进程或者线程争夺这些稀缺资源的时候，

  信号量用来保证他们合理地、秩序地使用这些资源，而不会陷入逻辑混乱。

  其中 ststem-V信号量、POSIX有名信号量是使用在进程间通信，而POSIX无名信号量使用在线程间通信.

  ststem-V信号量：是一种信号量的集合，它可以包含多个信号量元素的，

  而多个信号量元素是可以与信号量集合并存的。

POSIX有名信号量，POSIX无名信号量，多个资源是不可以并存的.

- **信号量的特点**

1.信号量是具有原子性操作的变量，是由电气特性直接提供的(CPU的电路直接提供支持的，硬件上支持的)。

2.信号量一定是一个非负数，信号量它所代表的意义是一个资源的数量，所以它一定是不为负数的。

如果某个信号量的值，变为了负数，则会阻塞等待，等待资源增加.

一些基本概念如下:

1，多个进程或线程有可能同时访问的资源（变量、链表、文件等等）称为共享资源，也叫临界资源（ critical resources ）。

2，访问这些资源的代码称为临界代码，这些代码区域称为临界区(critical zone)。

3，程序进入临界区之前必须要对资源进行申请，这个动作被称为P操作(减法).

4，程序离开临界区之后必须要释放相应的资源，这个动作被称为V操作(加法).

所有一起访问共同临界资源的进程都必须遵循以上游戏规则，否则大家就都乱套了.

```
==================================  信号量相关Shell命令  ================================

        查看信号量          : ipcs -s
        删除指定的信号量    : ipcrm -s SEM_ID 或者 ipcrm -S sem_key

```

二.相关API

- 申请信号量

```
功能 : 获取信号量的ID
NAME
```

```
       semget - get a System V semaphore set identifier

SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/sem.h>

       int semget(key_t key, int nsems, int semflg);
参数解析:
        参数1 key_t key  : 信号量的键值.
        参数2 int nsems  : 信号量的元素个数.
        参数3 int semflg : 访问权限.
                           IPC_CREAT : 不存在则创建
                           IPC_EXCL  : 信号量存在则报错
                           mode      : 信号量访问权限
返回值:
        成 功 : 返回信号量的ID.
        失 败 : 返回-1;
```

- 获取或者设置信号量的相关属性

```
==================================  semctl()函数解析  ================================
NAME
       semctl - System V semaphore control operations


SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/sem.h>
```

```
10
11      int semctl(int semid, int semnum, int cmd, ...);   //可变参函数
12
13  参数解析:
14          参数1 int semid  : 需要操作的信号量ID.
15          参数2 int semnum : 需要操作的信号量元素下标.(数组下标).
16          参数3 int cmd    : 宏定义.
17                             GETVAL 获取信号量的值
18                             SETVAL 设置信号量的值
19                             IPC_RMID 删除信号量.
20                             IPC_STAT 获取属性信息
21          该函数为可变参函数.根据参数3的变化而变化.
22  返回值:
23          该函数的返回值与参数3相关,
24          参数3 设置为 GETVAL 则返回 信号量的值;
25
26          失 败 : 返回-1;
27
28  代码使用简单示例:
29      int semctl(semid, 0, GETVAL);      //这时表示我要获取信号量序号(下标)为 0 的值, 成功返回信号量序号为 0 对应的值.
30      int semctl(semid, 0, SETVAL , 5); //这时表示我要设置信号量序号(下标)为 0 的值为 5. 成功则返回 0.
31      int semctl(semid, 0, IPC_RMID);    //这时表示我要删除信序号为0的信号量.
32
33
34  使用以上函数接口，需要注意以下几点:
35  1,这是一个变参函数，根据cmd 的不同，可能需要第四个参数，第四个参数是一个如下所示的联合体，用户必须自己定义:
36      union semun
37      {
38          int val;/*当cmd为SETVAL时使用*/
39          struct semid_ds *buf;      /*当cmd为IPC_STAT或IPC_SET时使用*/
```

```
        unsigned short  *array;   /*当cmd为GETALL或 SETALL时使用*/
        struct seminfo  *_buf;    /*当cmd为IPC_INFO时使用*/
    };

2,使用IPC_STAT 和 IPC_SET需要用到以下属性信息结构体:
    struct semid_ds
    {
        struct ipc_perm sem_perm;     /*权限相关信息*/
        time_t sem_otime;             /*最后一次semop( )的时间*/
        time_t sem_ctime;             /*最后一次状态改变时间*/
        unsigned shortsem_nsems;      /*信号量元素个数*/
    }
权限结构体如下:
    struct ipc_perm
    {
        key_t  __key;/*该信号量的键值 key */
        uid_t   uid;/*所有者有效UID*/
        gid_t   gid;/*所有者有效GID*/
        uid_t   cuid;/*创建者有效UID*/
        gid_t cgid;/*创建者有效GID岁/
        unsigned short mode;/*读写权限*/
        unsigned short _seq;/*序列号*/
    };

3,使用IPC_INFO时，需要提供以下结构体:
    struct seminfo
    {
        int semmap;  /*当前系统信号量总数*/
        int semmni;  /*产系统信号量个数最大值*/
        int semmns;  /*系统所有信号量元素总数最大值*/
```

```
        int semmnu;  /*信号量操作撤销结构体个数最大值*/
        int semmsl;  /*单个信号量中的信号量元素个数最大值*/
        int semopm;  /*调用semop()时操作的信号量元素个数最大值*/
        int semume;  /*单个进程对信号量执行连续撤销操作次数的最大值*/
        int semusz;  /*撤销操作的结构体的尺寸*/
        int semvmx;  /*信号量元素的值的最大值*/
        int semaem;  /*撤销操作记录个数最大值*/
    };
```

- 对临界资源进行申请，P操作，临界资源 -1. V操作 临界资源 +1

```
NAME
       semop, semtimedop - System V semaphore operations

SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/sem.h>

       int semop(int semid, struct sembuf *sops, size_t nsops);

参数解析 :
        参数1 int semid : 信号量ID.
        参数2 struct sembuf *sops : 结构体指针,结构体如下
            struct sembuf
            {
                  unsigned short sem_num;  /* 信号量数组下标,从 0 开始 */
                  short          sem_op;   /* 信号量的操作,加法(V操作)或减法(P操作) */
                  short          sem_flg;  /* 操作选项,SEM_UNDO,(还原信号量的值) */
                  //也就是说,你一开始设置了一个值,在做P操作或则V操作时,会改变信号量里所设置的值,Linux系统会保留其值,
```

```
                //设置这个宏定义之后,下次运行程序时,信号量的值会恢复原始你设置的值.(相当于复位).
                //该参数一般设置为0,,表示为标准行为
            }

        参数3 size_t nsops : 表示结构体数组元素个数,一般设置为1.

```

- 示例代码

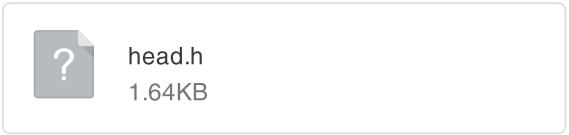
```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <errno.h>
/*
    演示信号量基本特点.
*/
int main(int argc, char const *argv[])
{
    //申请一个信号量
    int Sem_id = semget(12345, 1, IPC_CREAT | IPC_EXCL | 0777);
    if (Sem_id == -1)
    {
            if (errno == EEXIST)   //如果信号量已存在,则直接打开.
            {
                    Sem_id = semget(12345, 1, 0777);
            }
            else
```

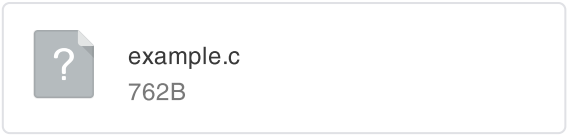
```
21          {
22                  perror("申请信号量失败\n");
23                  return -1;
24          }
25  }
26
27  //设置信号量初始值
28  int sem_ret = semctl(Sem_id, 0, SETVAL, 5);
29  if (sem_ret == -1)
30  {
31          printf("设置序号为 0 的信号量失败\n");
32  }
33
34  //查看信号量值是否设置成功
35  printf("信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));
36
37  //对申请的信号量进行P操作(减法操作)
38  struct sembuf var;
39  bzero(&var,sizeof(var));
40
41  var.sem_num = 0;   //表示要操作的信号量序号.
42  var.sem_op  = -1;  //表示申请一个资源. P操作, 一次要操作几个是可以自由决定的,不一定是1.也可以是其他的数据.
43  var.sem_flg = SEM_UNDO;
44
45  semop(Sem_id, &var, 1);
46  printf("操作后信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));
47
48  semop(Sem_id, &var, 1);
49  printf("操作后信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));
50
```

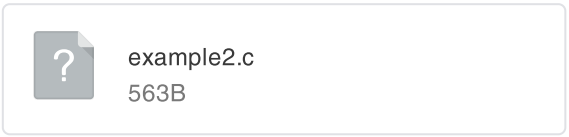
```
    semop(Sem_id, &var, 1);
    printf("操作后信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));

    semop(Sem_id, &var, 1);
    printf("操作后信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));

    semop(Sem_id, &var, 1);
    printf("操作后信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));

    semop(Sem_id, &var, 1);
    printf("操作后信号量中序号为 0 的值为 : %d\n", semctl(Sem_id, 0, GETVAL));

    return 0;
}

总结:
    1.信号量的值为 0 时,如果要继续进行 P(减法) 操作,会导致当前进程阻塞.
      信号量的值不可能为负数.假设你将该值设置为负数的话,该数据则不会生效.

    2.信号量在做 V(加法) 操作时,永远不会阻塞.
```

二值信号量

- 信号量pv操作封装代码.







POSIX有名信号量

我们试想一下，在我们程序运行的过程中会不会有特殊的意外发生导致我们的读写锁或者互斥锁变为死锁的情况。
那又有哪一种典型的情况我们的锁，会变为死锁.

这种有名信号量的名字由类似“/somename”这样的字符串组成，注意前面有一个正斜杠，
这样的信号量其实是一个特殊的文件，创建成功之后将会被放置在系统的一个特殊的虚拟文件系统/dev/shm 之中，
不同的进程间只要约定好一个相同的名字，他们就可以通过这种有名信号量来相互协调。

有名信号量跟system-V的信号量都是系统范畴的，在进程退出之后他们并不会自动消失，而需要手工删除并释放资源。

由于是有固定的名字，因此不同进程可以同时打开，适用于进程间同步互斥.

```
POSIX有名信号量的一般使用步骤是:
1，使用sem_open()来创建或者打开一个有名信号量。
2，使用sem_wait()和 sem_post()来分别进行P操作和V操作。
3，使用sem_close( )来关闭他。
4，使用sem_unlink()来删除他，并释放系统资源。

```

- 创建，打开一个POSIX有名信号量

```
NAME
       sem_open - initialize and open a named semaphore

SYNOPSIS
       #include <fcntl.h>           /* For O_* constants */
       #include <sys/stat.h>        /* For mode constants */
       #include <semaphore.h>

        sem_t *sem_open(const char *name, int oflag);
       sem_t *sem_open(const char *name,
                       int oflag,
                       mode_t mode,
                       unsigned int value);

       Link with -pthread.

参数解析：
    const char *name   : 信号量的名字.
    int oflag          :
                           O_CREATE 如果该名字对应的信号量不存在，则创建
                           O_EXCL   如果该名字对应的信号量已存在，则报错
    mode_t mode        :  八进制读写权限
    unsigned int value : 信号量的初始值.

返回值：
    成功 : 信号量指针     //这个指针不是普通的变量指针,这个指针是由操作系统维护的
                          //指向的是系统当中特定的位置。
```

```
    失败 : SEMM_FAILED;

    用法与OPEN函数类似.

```

- P，V操作

```
NAME
       sem_wait, sem_timedwait, sem_trywait - lock a semaphore

SYNOPSIS
       #include <semaphore.h>

       int sem_wait(sem_t *sem);    //等待资源 P操作 只-1  不够1则阻塞等待
       int sem_post(sem_t *sem);    //释放资源 V操作 只+1  此操作永远不会阻塞

参数解析:
    sem_t *sem : 信号量指针.

成功 : 0.
失败 : -1;

```

- 关闭，删除POSIX有名信号量

```
NAME
       sem_close - close a named semaphore
```

```
SYNOPSIS

        int sem_close(sem_t *sem);
        int sem _unlink(const char *name);

       Link with -pthread.

参数解析解析:
    sem_t *sem : 信号量指针.
    const char *name : 信号量名字;

返回值:
    成功 : 0;
    失败 : -1;
```

```
POSIX有名信号量使用示例代码
/*
    使用两个进程,通过共享内存收发数据
    进程1 发送数据给进程2. 输入完毕提醒进程2.
    进程2 等待进程1发送数据,收到数据则计算出字符串的长度.
*/


-------------------------example1
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
```

```c
#include <unistd.h>
#include <string.h>
#include <strings.h>
#include <errno.h>

#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/ipc.h>
#include <sys/shm.h>

sem_t *s;

void cleanup(int sig)
{
    sem_close(s);
    sem_unlink("data");
    exit(0);
}

int main(int argc, char **argv)
{
    signal(SIGINT, cleanup);

    // 创建共享内存
    int id = shmget(ftok(".", 1), 1024, IPC_CREAT|0666);
    char *addr = shmat(id, NULL, 0);
    bzero(addr, 1024);
```

```
43
44      // 创建一个有名POSIX信号量
45      s = sem_open("data", O_CREAT|O_EXCL, 0666, 0);
46      // 对方已经创建并初始化了该信号量
47      if(s == SEM_FAILED)
48      {
49            if(errno == EEXIST)
50            {
51                s = sem_open("data", O_CREAT);
52            }
53            else
54            {
55                 perror("sem_open() failed");
56                 exit(0);
57            }
58      }
59
60      // 不断输入数据，给对方计算长度
61      while(1)
62      {
63              fgets(addr, 1024, stdin);
64
65              // 通过信号量，通知对方可以计算了
66              sem_post(s);
67      }
68
69      return 0;
70  }
71
72
```

```
------------------------example2
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <strings.h>
#include <errno.h>

#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char **argv)
{
    // 创建共享内存
    int id = shmget(ftok(".", 1), 1024, IPC_CREAT|0666);
    char *addr = shmat(id, NULL, 0);

    // 创建/打开一个有名POSIX信号量
    sem_t *s;
    s = sem_open("data", O_CREAT|O_EXCL, 0666, 0);

    // 对方已经创建并初始化了该信号量
    if(s == SEM_FAILED)
    {
```

```
        if(errno == EEXIST)
        {
            s = sem_open("data", O_CREAT);
        }
        else
        {
             perror("sem_open() failed");
           exit(0);
        }
    }

    // 静静地等待信号量的提示....
    while(1)
    {
            // 以下函数会引起进程睡眠。。。
            if(sem_wait(s) == -1)
            {
                    break;
            }
            printf("%lu\n", strlen(addr));
    }
    sem_close(s);
    return 0;
}

```

POSIX无名信号量

一.概念

如果我们要解决的是一个进程内部的线程间的同步互斥,那么也许不需要使用有名信号量，

因为这些线程共享同一个内存空间，我们可以定义更加轻量化的、基于内存的(不在任何文件系统内部）无名信号量来达到目的。

无名信号量由于存在于线程的共同内存当中，因此可以非常方便的与进程内的各条线程同步互斥，一旦进程退出信号量也将消失

```
这种信号量的使用步骤是:
1，在这些线程都能访问到的区域定义这种变量（比如全局变量），类型是sem_t.
2，在任何线程使用它之前，用sem_init()初始化他。
3，使用sem_wait( )和 sem_post()来分别进行P、V操作。
4，不再需要时，使用sem_destroy()来销毁他。

    其中sem_wait()和 sem_post()跟 POSIX有名信号量是一样的，初始化和销毁函数接口有所不同。
```

- 初始化、销毁POSIX无名信号量

```
头文件
#include <semaphore.h>

原型:
    int sem_init(sem_t *sem, int pshared, unsigned int value);
    int sem_destroy(sem_t *sem);sem

参数解析:

        sem_t *sem  : 信号量指针;
        int pshared : 该信号量的作用范围: 0为线程间，非0为进程间;
                      无名信号量一般用在进程内的线程间，因此pshared参数一般都为0。
```

```
                当将此种信号量用在进程间时，必须将他定义在各个进程都能访问的地方——比如共享内存之中。

        unsigned int value : 信号量初始值
返回值
        成功 : 返回0
        失败 : 返回-1

```

```
使用示例:
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <strings.h>
#include <errno.h>

#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

#include <pthread.h>
#include <semaphore.h>

// 定义一个POSIX无名信号量
sem_t s;

void *routine(void *arg)
```

```c
{
    char *buf = (char *)arg;

    // 计算buf中的字符串的长度
    while(1)
    {
            // 静静等待对方的数据准备好。。。
            // P操作
            sem_wait(&s);
            printf("%lu\n", strlen(buf));
    }
}

int main(int argc, char **argv)
{
    char buf[100];
    sem_init(&s, 0/* 零代表线程间使用，非零是进程间使用*/, 0/*初始值*/);

    pthread_t tid;
    pthread_create(&tid, NULL, routine, (void *)buf);

    // 不断输入数据到buf
    while(1)
    {
            fgets(buf, 100, stdin);

            // 使用信号量通知对方可以开始计算了
            // V操作
            sem_post(&s);
    }
```

```
    return 0;
}

```

信号量总结：

```
对于我们接触过的三种信号量:system-V信号量和POSIX信号量(named-sem 和 unnamed-sem)

    1，sys-V信号量较古老，语法艰涩。POSIX信号量简单，轻量。
    2，sys-V信号量可以对代表多种资源的多个信号量元素同一时间进行原子性的P/V操作，POSIX信号量每次只能操作一个信号量。
    3，sys-V信号量和named-sem是系统范围的资源，进程消失之后继续存在，而unnamed-sem是进程范围的资源，随着进程的退出而消失。
    4，sys-V信号量的P/V操作可以对信号量元素加减大于1的数值，而POSIX信号量每次P/V操作都是加减1。
    5，sys-V信号量甚至还支持撤销操作——一个进程对sys-V信号量进行P/V操作时可
        以给该操作贴上需要撤销的标识，那么当进程退出之后，系统会自动撤销那些做了标识的操作。而 POSIX信号没有此功能。
    6，sys-V信号量和 named-sem适合用在进程间同步互斥，而unamed-sem适合用在线程间同步互斥。


总的来说，system-V的信号量功能强大，强大到臃肿啰嗦，如果在现实工作中不需要那些高级功能，建议使用接口清晰、逻辑简单的POSIX信号量。
```

作业:

一.基于消息队列的通信方式实现一个服务器P1进程 / 客户端 p2进程模式系统：

1.服务器从消息队列读取需要计算的两个整数，求和后把结果通过消息队列返回给客户端。

2.客户端从键盘读取两个整数通过消息队列发送给服务器，接收服务器的计算结果后显示出来

3.（选做）要求服务器能够同时处理多个客户端的请求.

二.用信号量去协调使用 共享内存 实现双向通信.