Linux IPC 之信号量

1.1. 信号量

信号量（也叫信号灯）是一种用于提供不同进程间或一个给定进程的不同线程间同步手段的原语。信号量是进程/线程同步的一种方式，有时候我们需要保护一段代码，使它每次只能被一个执行进程/线程运行，这种工作就需要一个二进制开关；有时候需要限制一段代码可以被多少个进程/线程执行，这就需要用到关于计数信号量。信号量开关是二进制信号量的一种逻辑扩展，两者实际调用的函数都是一样。

信号量分为以下三种。

1、System V 信号量，在内核中维护，可用于进程或线程间的同步，常用于进程的同步。

2、Posix 有名信号量，一种来源于 POSIX 技术规范的实时扩展方案（POSIX Realtime Extension）,可用于进程或线程间的同步，常用于线程。

3、Posix 基于内存的信号量，存放在共享内存区中，可用于进程或线程间的同步。

为了获得共享资源进程需要执行下列操作：

（1）测试控制该资源的信号量。

（2）若信号量的值为正，则进程可以使用该资源。进程信号量值减 1，表示它使用了一个资源单位。此进程使用完共享资源后对应的信号量会加 1。以便其他进程使用。

（3）若信号量的值为 0，则进程进入休息状态，直至信号量值大于 0。进程被唤醒，返回第（1）步。

为了正确地实现信号量，信号量值的测试值的测试及减 1 操作应当是原子操作（原子操作是不可分割的，在执行完毕前不会被任何其它任务或事件中断）。为此信号量通常是在内核中实现的。

- **System V** IPC 机制：信号量。

函数原型：

```
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/types.h>
int semget(key_t key,int nsems,int flag);
int semop(int semid,struct sembuf *sops,size_t num_sops);
int semctl(int semid, int semnum, int cmd, …);
```

函数 `semget` 创建一个信号量集或访问一个已存在的信号量集。返回值：成功时，返回一个称为信号量标识符的整数，semop 和 semctl 会使用它；出错时，返回-1.

参数 key 是唯一标识一个信号量的关键字，如果为 IPC_PRIVATE(值为 0，创建一个只有创建者进程才可以访问的信号量，通常用于父子进程之间；非 0 值的 key(可以通过 ftok 函数获得)表示创建一个可以被多个进程共享的信号量；

参数 nsems 指定需要使用的信号量数目。如果是创建新集合，则必须制定 nsems。如果引用一个现存的集合，则将 nsems 指定为 0.

参数 flag 是一组标志，其作用与 open 函数的各种标志很相似。它低端的九个位是

该信号量的权限，其作用相当于文件的访问权限。此外，它们还可以与键值 IPC_CREAT 按位或操作，以创建一个新的信号量。即使在设置了 IPC_CREAT 标志后给出的是一个现有的信号量的键字，也并不是一个错误。我们也可以通过 IPC_CREAT 和 IPC_EXCL 标志的联合使用确保自己将创建出一个新的独一无二的信号量来，如果该信号量已经存在，就会返回一个错误。

函数 semop 用于改变信号量对象中各个信号量的状态。返回值：成功时，返回 0；失败时，返回-1.

参数 semid 是由 semget 返回的信号量标识符。

参数 sops 是指向一个结构体数组的指针。每个数组元素至少包含以下几个成员：

struct sembuf{

short sem_num; //操作信号量在信号量集合中的编号，第一个信号量的编号是 0。

short sem_op; //sem_op 成员的值是信号量在一次操作中需要改变的数值。通常只会用到两个值，一个是-1，也就是 p 操作，它等待信号量变为可用；一个是+1，也就是 v 操作，它发送信号通知信号量现在可用。

short sem_flg; //通常设为：SEM_UNDO，程序结束，信号量为 semop 调用前的值。

};

1.参数 nops 为 sops 指向的 sembuf 结构体数组的大小。

函数 semctl 用来直接控制信号量信息。函数返回值：成功时，返回 0；失败时，返回-1.

2.参数 semid 是由 semget 返回的信号量标识符。

3.参数 semnum 为集合中信号量的编号，当要用到成组的信号量时，从 0 开始。一般取值为 0，表示这是第一个也是唯一的一个信号量。

4. 参数 cmd 为执行的操作。通常为：IPC_RMID（立即删除信号集，唤醒所有被阻塞的进程）、GETVAL（根据 semun 返回信号量的值，从 0 开始，第一个信号量编号为 0）、SETVAL（根据 semun 设定信号的值，从 0 开始，第一个信号量编号为 0）、GETALL（获取所有信号量的值，第二个参数为 0，将所有信号的值存入 semun.array 中）、SETALL（将所有 semun.array 的值设定到信号集中，第二个参数为 0）等。

参数…是一个 union semun（需要由程序员自己定义），它至少包含以下几个成员：

```
union semun{
    int val;                    /* Value for SETVAL */
    struct semid_ds *buf;       /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array;      /* Array for GETALL, SETALL */
};
```

通常情况仅使用 val，给 val 赋值为 1

The semid_ds data structure is defined in <sys/sem.h> as follows:

```
struct semid_ds {
    struct ipc_perm sem_perm;   /* Ownership and permissions */
    time_t          sem_otime;  /* Last semop time */
    time_t          sem_ctime;  /* Last change time */
    unsigned long   sem_nsems;  /* No. of semaphores in set */
};
```

The ipc_perm structure is defined as follows (the highlighted fields are settable using IPC_SET):

```
struct ipc_perm {
    key_t          __key; /* Key supplied to semget(2) */
    uid_t          uid;   /* Effective UID of owner */
    gid_t          gid;   /* Effective GID of owner */
    uid_t          cuid;  /* Effective UID of creator */
    gid_t          cgid;  /* Effective GID of creator */
    unsigned short mode;  /* Permissions */
    unsigned short __seq; /* Sequence number */
};
```

示例：有亲缘关系的信号量进程间通信

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <string.h>

union semun              //必须重写这个共用体
{
    int val;
    struct semid_ds *buf;
    unsigned short* array;
};
int main()
{
    int semid = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT);//创建信号量集
    if(semid == -1)
    {
        perror("semget error");
        exit(-1);
    }
    if(fork() == 0)      //表示子进程
    {
        struct sembuf sem;          //定义信号量结构体
        memset(&sem, 0, sizeof(struct sembuf));
        sem.sem_num = 0;            //信号量的编号,第一个为 0
        sem.sem_op = 1;             //+1  表示执行 V 操作,生产产品
```

```
            sem.sem_flg = 0;                //或写 SEM_UNDO
            union semun arg;
            arg.val = 0;                    //初始化数据
            semctl(semid, 0, SETALL, arg);  //将数据全部设置到信号量集里面去,相当于
公共数据
            while(1)
            {
                semop(semid, &sem, 1); //执行指定的 V 操作,表示生产产品
                printf("productor total number:%d\n", semctl(semid, 0, GETVAL));//获得公
共值
                sleep(1);
            }
        }
        else
        {
            sleep(2);                       //先让子进程有时间生产
            struct sembuf sem;              //定义信号量结构体
            memset(&sem, 0, sizeof(struct sembuf));
            sem.sem_num = 0;                //信号量的编号,第一个为 0
            sem.sem_op = -1;                //-1  表示执行 P 操作,消费产品
            sem.sem_flg = 0;                //或写 SEM_UNDO
            while(1)
            {
                semop(semid, &sem, 1);//执行指定的 P 操作消费产品
                printf("costomer total number:%d\n", semctl(semid, 0, GETVAL));//获得公
共值
                sleep(2);
            }
        }
}
```

示例：没有亲缘关系的生产者消费者问题

生产者源码：

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
void init();      //initlization semaphore
void del();       //delete semaphore
int sem_id;
int main(int argc, char *argv[])
{
```

```
    struct sembuf sops[2];
    /*set operate way for semaphore*/
    sops[0].sem_num = 0;    //第一个信号的编号，表示生产了几个产品
    sops[0].sem_op = 1;     //进行 V 操作
    sops[0].sem_flg = 0;    //或写为 SEM_UNDO
    sops[1].sem_num = 1;    //第二个信号编号，表示还可以生产几个产品
    sops[1].sem_op = -1;    //进行 P 操作
    sops[1].sem_flg = 0;    //或写为 SEM_UNDO
    init();     //初始化操作
    printf("this is a producor\n");
    while(1)
    {
        printf("\n\nbefore produce\n");
        printf("productor number is %d\n", semctl(sem_id, 0, GETVAL));
        printf("space number is %d\n", semctl(sem_id, 1, GETVAL));
        semop(sem_id, (struct sembuf*)&sops[1], 1);
        printf("now producing .....\n");
        semop(sem_id, (struct sembuf*)&sops[0], 1);
        printf("space number is %d\n", semctl(sem_id, 1, GETVAL));
        printf("productor number is %d\n", semctl(sem_id, 0, GETVAL));

        sleep(2);
    }
    del();
}
void init()
{
    int ret;
    unsigned short sem_array[2];
    union semun
    {
        int val;
        struct semid_ds *buf;
        unsigned short *array;
    }arg;
    sem_id = semget((key_t)1234, 2, IPC_CREAT|0644);/*get semaphore include two
sem*/

    /*set sem's vaule*/
    sem_array[0] = 0;
    sem_array[1] = 10;
    arg.array = sem_array;
    ret = semctl(sem_id, 0, SETALL, arg);  //将所有 semun.array 的值设定到信号集中，第
二个参数为 0
```

```
    if(ret == -1)
    {
        printf("SETALL failed (%d)\n", errno);
    }
    printf("productor init is %d\n", semctl(sem_id, 0, GETVAL));
    printf("space init is %d\n\n", semctl(sem_id, 1, GETVAL));
}
void del()
{
    semctl(sem_id, IPC_RMID, 0);    //删除信号集
}
```

消费者源码：

```
#include<unistd.h>
#include  <stdlib.h>
#include  <stdio.h>
#include  <sys/types.h>
#include  <sys/ipc.h>
#include  <sys/sem.h>
#include  <errno.h>

void init();
int sem_id;

int main(int argc, char *argv[])
{
    struct sembuf sops[2];
    /*set operate way for sem*/
    sops[0].sem_num = 0;
    sops[0].sem_op = -1;    //P 操作，表示有多少个产品可以消费（相当于生产了多少个产品）
    sops[0].sem_flg = 0;
    sops[1].sem_num = 1;
    sops[1].sem_op = 1;     //V 操作，表示还可以生成的产品数
    sops[1].sem_flg = 0;
    init();
    printf("this is a customer\n");
    while(1)
    {
        printf("\n\nbefore consume\n");
        printf("customer number is %d\n", semctl(sem_id, 0, GETVAL));
        printf("space number is %d\n", semctl(sem_id, 1, GETVAL));
        semop(sem_id, &sops[0], 1);
        printf("now consume .....\n");
        semop(sem_id, &sops[1], 1);
```

```
            printf("space number is %d\n", semctl(sem_id, 1, GETVAL));
            printf("customer number is %d\n", semctl(sem_id, 0, GETVAL));
            sleep(1);
        }
    }
    void init()
    {
        sem_id = semget((key_t)1234, 2, IPC_CREAT|0644);/*get semaphore include two
    sem*/
    }
```

在生产者源码里，首先用函数 semctl()初始化信号量集合 sem_id，它包含两个信号，分别表示生产的数量和空仓库的数量，那么在消费者的进程中用相同的 key 值就会得到该信号量集合；实现两个进程之间的通信。

在主函数里，设定对两个信号量的 PV 操作，然后在各自的进程中对两个信号进行操作。

（1）如果只运行生产者进程，则生产 10 个之后，该进程就会因为在得不到空仓库资源而阻塞，这个时候运行消费者进程，阻塞就会被解除；

（2）如果先运行生产者进程，生产几个产品之后，关闭该进程，则运行消费者进程，当消费完生产的产品后，该进程就会因为在得不到产品资源而阻塞，这个时候运行生产者进程，阻塞就会被解除；

（3）如果同时运行两个进程，由于消费比生产快，因此消费者每次都要等待生产者生产产品之后才能消费；

在每次运行程序之前，一定要先运行生产者进程先初始化信号量。