



# 实验二、线程同步与通信

## 一、实验目的

- 1、掌握Linux下线程的概念；
- 2、了解Linux线程同步与通信的主要机制；
- 3、通过信号灯操作实现线程间的同步与互斥。



## 二、实验内容

通过Linux多线程与信号灯机制，设计并实现计算机线程与I/O线程共享缓冲区的同步与通信。

程序要求：两个线程，共享公共变量a

线程1负责计算（1到100的累加，每次加一个数）

线程2负责打印（输出累加的中间结果）



### 三、预备知识

## 1、Linux下的信号灯及其P、V操作

在Linux中信号灯是一个数据集合，可以单独使用这一集合的每个元素。

有关的系统调用命令：

- 1) semget：返回一个被内核指定的整型的信号灯索引。
- 2) semop：执行对信号灯集的操作
- 3) semctl：执行对信号灯集的控制操作。

- 信号灯的定义：

数据结构的原型是semid\_ds，在linux/sem.h中定义：

```
struct semid_ds{  
    struct ipc_permsem_perm; /*permissions.. seeipc.h*/  
    time_t sem_otime; /*last semop time*/  
    time_t sem_ctime; /*last change time*/  
    struct sem*sem_base; /*ptr to first semaphore in array*/  
    struct wait_queue *eventn;  
    struct wait_queue *eventz;  
    struct sem_undo*undo; /*undo requests on this array*/  
    ushort sem_nsems; /*no. of semaphores in array*/  
};
```

## ● 信号量的创建：

使用系统调用semget()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。

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

返回值：如果成功，则返回信号量集的IPC标识符。如果失败，则返回-1.

semflg:IPC\_CREAT | 0666

nsems:信号灯的个数

● 信号量的操作：系统调用semop();

调用原型：int semop(int semid, struct sembuf  
\*sops, unsigned nsops);

返回值：0，如果成功。-1，如果失败

```
struct sembuf {  
    ushort sem_num; /*semaphore index in array*/  
    short sem_op; /*semaphore operation*/  
    short sem_flg; /*operation flags*/  
}
```

sem\_num 将要处理的信号量的下标。

sem\_op 要执行的操作。

sem\_flg 操作标志, IPC\_NOWAIT

## ● P操作

```
void P(int semid, int index)
{
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;
    sem.sem_flg = 0; //操作标记: 0或IPC_NOWAIT等
    semop(semid, &sem, 1); //1:表示执行命令的个数
    return;
}
```

## ● V操作

```
void V(int semid, int index)
```

```
{
```

```
    struct sembuf sem;
```

```
    sem.sem_num = index;
```

```
    sem.sem_op = 1;
```

```
    sem.sem_flg = 0;
```

```
    semop(semid, &sem, 1);
```

```
    return;
```

```
}
```

- 信号量的赋值：系统调用semctl()

原型：int semctl(int semid, int semnum, int cmd, union semun arg);

返回值：如果成功，则为一个正数。

参数cmd中可以使用的命令如下：

- IPC\_RMID将信号量集从内存中删除。
- SETALL设置信号量集中的所有的信号量的值。
- SETVAL设置信号量集中一个单独的信号量的值。

arg. val=1;

semctl(semid, 1, SETVAL, arg)



## 2、线程

### 1) 线程创建

```
pthread_create(pthread_t *thread, pthread_attr_t  
              *attr,  
              void *(*start_routine)(void *), void *arg);
```

2) pthread\_join(pthread\_t th, void \*\*thread\_retrun);

作用：挂起当前线程直到由参数th指定的线程被终止为止。

### 3、编辑、编译、调试

\$vi

\$cc -o test -g test.c -lpthread

\$gdb



## 四、实验指导

### 程序结构

#include 头文件pthread.h、sys/types.h、  
linux/sem.h等

P、V操作的函数定义：

void P(int semid, int index)

void V(int semid, int index)

信号灯、线程句柄定义：

int semid; pthread\_t p1, p2;

线程执行函数定义： void \*subp1(); void \*subp2();



```
void *subp1()  
{  
    for (.....) {  
        P(.....);           主函数: main()  
        打印;               {  
        V(...);              创建信号灯;  
    }                      信号灯赋初值;  
    return;                创建两个线程subp1、subp2;  
}  
}
```

subp2负责计算，如何定义？