第九章：多任务互斥和同步

9.1 互斥和同步概述

在多任务操作系统中，同时运行的多个任务可能都需要访问/使用同一种资源
多个任务之间有依赖关系，某个任务的运行依赖于另一个任务
同步和互斥就是用于解决这两个问题的。

互斥:

一个公共资源同一时刻只能被一个进程或线程使用，多个进程或线程不能同时使用公共资源。POSIX 标准中进程和线程同步和互斥的方法,主要有信号量和互斥锁两种方式。

同步:

两个或两个以上的进程或线程在运行过程中协同步调，按预定的先后次序运行。

9.2 互斥锁

9.2.1 互斥锁的概念

mutex 是一种简单的加锁的方法来控制对共享资源的访问，mutex 只有两种状态,即上锁(lock)和解锁(unlock)。
在访问该资源前，首先应申请 mutex，如果 mutex 处于 unlock 状态，则会申请到 mutex 并立即 lock；
如果 mutex 处于 lock 状态，则默认阻塞申请者。
unlock 操作应该由 lock 者进行。

9.2.2 初始化互斥锁

mutex 用 pthread_mutex_t 数据类型表示，在使用互斥锁前,必须先对它进行初始化。

静态分配的互斥锁：

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

动态分配互斥锁:

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);在所有使用过此互斥锁的线程都不再需要使用时候，应调用
```

销毁互斥锁

pthread_mutex_destroy 销毁互斥锁。

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex,const pthread_mutexattr_t *attr);
```

功能：

初始化一个互斥锁。

参数：

mutex：互斥锁地址。

attr：互斥锁的属性，NULL 为默认的属性。

返回值：

成功返回 0，失败返回非 0。

9.2.3 互斥锁上锁

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

功能：

对互斥锁上锁，若已经上锁，则调用者一直阻塞到互斥锁解锁。

参数：

mutex：互斥锁地址。

返回值：

成功返回 0，失败返回非 0。

9.2.4 互斥锁上锁 2

```
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

功能：

对互斥锁上锁，若已经上锁，则上锁失败，函数立即返回。

参数：

mutex：互斥锁地址。

返回值：

成功返回 0，失败返回非 0。

9.2.5 互斥锁解锁

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t * mutex);
```

功能：

对指定的互斥锁解锁。

参数：

mutex：互斥锁地址。

返回值：

成功返回 0,失败返回非 0。

9.2.6 销毁互斥锁

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

功能：

销毁指定的一个互斥锁。

参数：

mutex：互斥锁地址。

返回值：

成功返回 0，失败返回非 0。

例：01_pthread_mutex.c 互斥锁实现模拟打印机

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void printer(char *str)
{
    pthread_mutex_lock(&mutex);
    while(*str!='\0')
    {
        putchar(*str);
        fflush(stdout);
        str++;
        sleep(1);
    }
    printf("\n");
    pthread_mutex_unlock(&mutex);
}

void *thread_fun_1(void *arg)
{
    char *str = "hello";
    printer(str);
}

void *thread_fun_2(void *arg)
{
```

```
    char *str = "world";
    //pthread_mutex_unlock(&mutex);
    printer(str);
}

int main(void)
{
    pthread_t tid1, tid2;

    pthread_mutex_init(&mutex, NULL);
    pthread_create(&tid1, NULL, thread_fun_1, NULL);
    pthread_create(&tid2, NULL, thread_fun_2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    return 0;
}
```

9.3 信号量

9.3.1 信号量的概念

信号量广泛用于进程或线程间的同步和互斥，信号量本质上是一个非负的整数计数器，它被用来控制对公共资源的访问。

编程时可根据操作信号量值的结果判断是否对公共资源具有访问的权限，当信号量值大于 0 时，则可以访问，否则将阻塞。

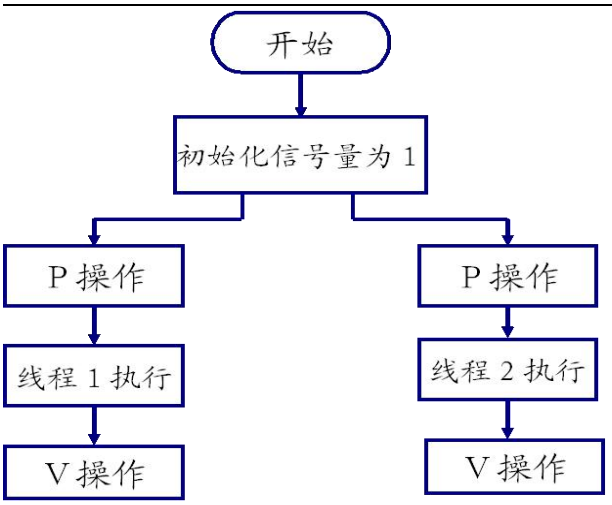
P V 原语是对信号量的操作，一次 P 操作使信号量 sem 减 1 ，一次 V 操作使信号量 sem 加 1 。

信号量主要用于进程或线程间的同步和互斥这两种典型情况。

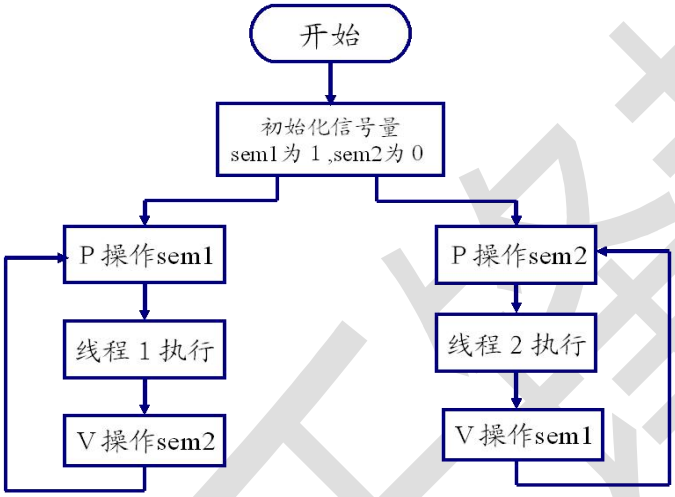
1、若用于互斥，几个进程（或线程）往往只设置一个信号量。

2、若用于同步操作，往往会设置多个信号量，并且安排不同的初始值，来实现它们之间的执行顺序。

信号量用于互斥



信号量用于同步



9.3.2 信号量的操作

9.3.2.1 信号量的初始化

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared,unsigned int value);
```

功能：

创建一个信号量并初始化它的值。

参数：

sem：信号量的地址。

pshared：等于 0，信号量在线程间共享；不等于 0，信号量在进程间共享。

value：信号量的初始值。

返回值：

成功返回 0，失败返回-1。

9.3.2.2 信号量 P 操作

#include <semaphore.h>

int sem_wait(sem_t *sem);

功能：

将信号量的值减 1，若信号量的值小于等于 0，此函数会引起调用者阻塞。

参数：

sem：信号量地址。

返回值：

成功返回 0，失败返回-1。

#include <semaphore.h>

int sem_trywait(sem_t *sem);

功能：

将信号量的值减 1，若信号量的值小于等于 0，则对信号量的操作失败，函数立即返回。

参数：

sem：信号量地址。

返回值：

成功返回 0，失败返回-1。

9.3.2.3 信号量的 V 操作

#include <semaphore.h>

int sem_post(sem_t *sem);

功能：

将信号量的值加 1 并发出信号唤醒等待线程。

参数：

sem：信号量地址。

返回值：

成功返回 0，失败返回-1。

9.3.2.4 获取信号量的计数值

#include <semaphore.h>

int sem_getvalue(sem_t *sem, int *sval);

功能：

获取 sem 标识的信号量的值，保存在 sval 中。

参数：

sem：信号量地址。

sval：保存信号量值的地址。

返回值：

成功返回 0，失败返回-1。

9.3.2.5 信号量的销毁

```
#include <semaphore.h>
int sem_destroy(sem_t *sem);
```

功能：

删除 sem 标识的信号量。

参数：

sem：信号量地址。

返回值：

成功返回 0，失败返回-1。

例：02_semphore_1.c　信号量实现互斥功能，模拟打印机

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>

sem_t sem;

void printer(char *str)
{
    sem_wait(&sem);
    while(*str)
    {
        putchar(*str);
        fflush(stdout);
        str++;
        sleep(1);
    }
    sem_post(&sem);
}

void *thread_fun1(void *arg)
```

```
{
    char *str1 = "hello";
    printer(str1);
}

void *thread_fun2(void *arg)
{
    char *str2 = "world";
    printer(str2);
}

int main(void)
{
    pthread_t tid1, tid2;

    sem_init(&sem, 0, 1);
    pthread_create(&tid1, NULL, thread_fun1, NULL);
    pthread_create(&tid2, NULL, thread_fun2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    return 0;
}
```

例：02_semphore_2.c 验证信号量实现同步功能

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

sem_t sem_g,sem_p;    //定义两个信号量
char ch = 'a';

void * pthread_g(void *arg)   //此线程改变字符 ch 的值
{
    while(1)
    {
```

```
            sem_wait(&sem_g);
            ch++;
            sleep(1);
            sem_post(&sem_p);
        }
}
void * pthread_p(void *arg)   //此线程打印 ch 的值
{
        while(1)
        {
            sem_wait(&sem_p);
            printf("%c",ch);
            fflush(stdout);
            sem_post(&sem_g);
        }
}

int main(int argc, char *argv[])
{
        pthread_t tid1,tid2;
        sem_init(&sem_g, 0, 0);     //初始化信号量
        sem_init(&sem_p, 0, 1);

        pthread_create(&tid1,NULL,pthread_g,NULL);
        pthread_create(&tid2,NULL,pthread_p,NULL);

        pthread_join(tid1,NULL);
        pthread_join(tid2,NULL);
        return 0;
}
```