


第8章 线程间的同步机制
——条件变量

主　讲：王小银

条件变量基本原理

- 条件变量类似于if语句，有“真”、“假”两种状态。
- 在条件变量的使用过程中一个线程等待条件为“真”，另一个线程在使用完临界资源之后将条件设置为“真”，唤醒阻塞在等待条件变量为“真”的线程，执行其任务。
- 条件变量一般需要和互斥锁配合使用实现对资源的互斥访问。

初始化条件变量

函数名称	pthread_cond_init
函数功能	初始化条件变量
头文件	#include <pthread.h>
函数原型	int pthread_cond_init(pthread_cond_t *cv, const pthread_condattr_t *cattr);
参数	cv：指向要初始化的条件变量的指针； cattr：指向条件变量属性的指针，指定条件变量的一些属性，如果为NULL则表明使用默认属性初始化该条件变量。
返回值	0：成功； 非0：失败。

阻塞等待条件变量（1）

函数名称	pthread_cond_wait
函数功能	等待条件变量被设置
头文件	#include <pthread.h>
函数原型	int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
参数	cond：　　需要等待的条件变量； mutex：　　与条件变量相关的互斥锁。
返回值	0：　　成功； 非0：　　失败。

阻塞等待条件变量（2）

函数名称	pthread_cond_timedwait
函数功能	在指定的时间之内阻塞等待条件变量
头文件	#include <pthread.h>
函数原型	int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
参数	cond：需要等待的条件变量； mutex：与条件变量绑定的互斥锁； abstime：struct timespec类型的指针变量，传入超时时间，该变量是一个绝对时间，即以从1970-01-01 00:00:00 以来的秒数来表示。
返回值	0：成功； 非0：失败。

通知等待该条件变量的线程

函数名称	pthread_cond_signal
函数功能	唤醒一个等待线程
头文件	#include <pthread.h>
函数原型	int pthread_cond_signal(pthread_cond_t *__cond);
参数	cond：　　需要通知的条件变量的指针。
返回值	0：　　成功； 非0：　　失败。

通知等待该条件变量的线程

函数名称	pthread_cond_ broadcast
函数功能	唤醒所有等待线程
头文件	#include <pthread.h>
函数原型	int pthread_cond_ broadcast (pthread_cond_t *__cond);
参数	cond：　　需要广播通知的条件变量的指针。
返回值	0：　　成功； 非0：　　失败。

销毁条件变量

函数名称	**pthread_cond_destroy**
函数功能	销毁条件变量
头文件	#include <pthread.h>
函数原型	int pthread_cond_destroy(pthread_cond_t *__cond);
参数	cond：　　指向条件变量对象的指针。
返回值	0：　　成功； 非0：　　失败。

示例程序

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

pthread_mutex_t mutex;     // 互斥锁
pthread_cond_t empty;      // 为空的条件变量
pthread_cond_t notempty;   // 非空的条件变量
char buf[32];
void *producer(void *arg)
{    while(1)
    {  printf("producer is runing!\n");
       pthread_mutex_lock(&mutex);
       pthread_cond_wait(&empty, &mutex);
       memcpy(buf, "Hello Linux", 11);            // 写入数据
       pthread_cond_signal(&notempty);        //非空
       pthread_mutex_unlock(&mutex);
    }
    return 0;
}
```

```
void *consume(void *arg)
{
    while(1)
    {
       printf("consume is runing!\n");
       pthread_mutex_lock(&mutex);
       pthread_cond_wait(&notempty, &mutex);
                                  // 等待缓存区不为空
       printf("recv data : %s\n", buf);
       memset(buf, 0, 32);
       sleep(1);
       pthread_cond_signal(&empty);          // 为空
       pthread_mutex_unlock(&mutex);
    }

    return 0;
}
```

示例程序

```
int main(int argc, char *argv[])
{   pthread_t thid1, thid2;
    int *ret1, *ret2;
    pthread_mutex_init(&mutex, NULL);       // 初始化互斥锁
    pthread_cond_init(&empty, NULL);        // 初始化为空时的条件变量
    pthread_cond_init(&notempty, NULL);     // 初始化不为空时的条件变量
    pthread_create(&thid1, NULL, producer, NULL);
    pthread_create(&thid2, NULL, consume, NULL);

    sleep(1);
    pthread_cond_signal(&empty);            // 为空

    pthread_join(thid1, (void **)&ret1);
    pthread_join(thid2, (void **)&ret2);
    pthread_mutex_destroy(&mutex);    // 销毁互斥锁
    pthread_cond_destroy(&empty);     // 销毁条件变量
    pthread_cond_destroy(&notempty);  // 销毁条件变量
    return EXIT_SUCCESS;
}
```

运行结果：

```
root@ubuntu:~$ ./exp_cond
consume is runing!
producer is runing!
producer is runing!
recv data : Hello Linux
consume is runing!
producer is runing!
recv data : Hello Linux
consume is runing!
producer is runing!
recv data : Hello Linux
consume is runing!
producer is runing!
recv data : Hello Linux
……
```

谢谢大家！