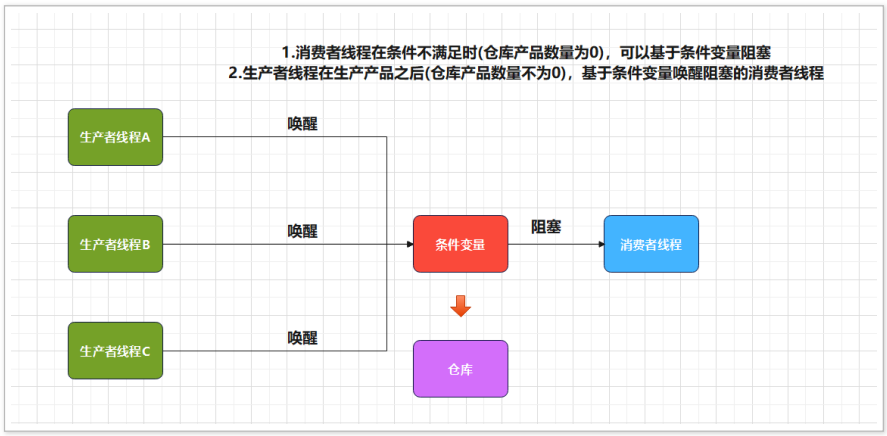
10.3 条件变量_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 10.3 条件变量涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

- 不足:
    - 主线程 (消费者线程) 需要不断查询是否有产品可以消费, 如果没有产品可以消费，也在运行程序，包括获得互斥锁、判断条件、释放互斥锁，非常消耗 cpu 资源
- 条件变量 允许一个线程就某个共享变量的状态变化通知其他线程, 并让其他线程等待这一通知



- 条件变量的本质为 pthread_cond_t 类型的变量, 其他线程可以阻塞在这个条件变量上, 或者唤醒阻塞在这个条件变量上的线程
- 条件变量的初始化分为 静态初始化 与动态初始化
    - 静态初始化

pthread_cond_t cond = PTHREAD_COND_INITALIZER;

    - 动态初始化
        - pthread_cond_init 函数

          函数头文件 #include <pthread.h>

          函数原型 int pthread_cond_init(pthread_cond_t *restrict cond,

          const pthread_condattr_t *restrict attr);

          函数功能 初始化条件变量

          函数参数
            - cond：条件变量指针
            - attr：条件变量属性

          函数返回值
            - 成功：返回 0
            - 失败：返回错误码
        - pthread_cond_destroy

          函数头文件 #include <pthread.h>

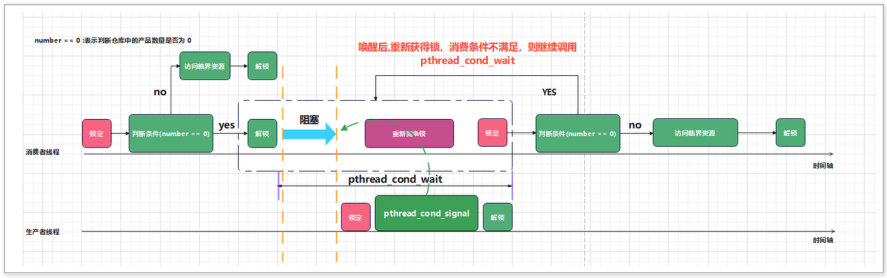
          函数原型 int pthread_cond_destroy(pthread_cond_t *cond);

          函数功能 销毁条件变量

          函数参数 cond：条件变量指针

          函数返回值
            - 成功：返回 0

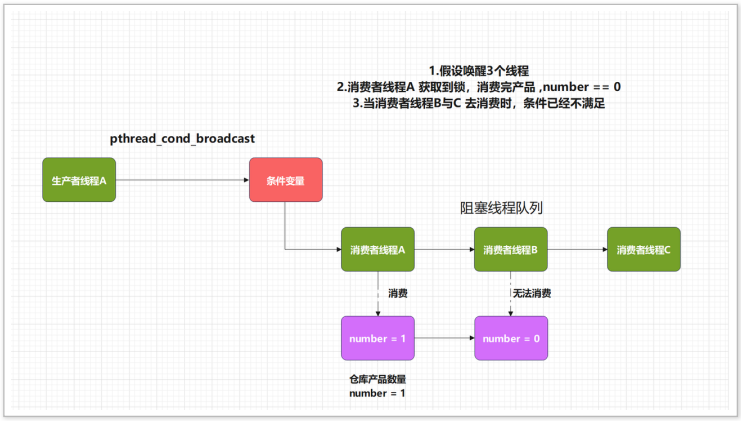
- 失败：返回错误码



- step 1：消费者线程判断消费条件是否满足（仓库是否有产品），如果有产品可以消费，则可以正常消费产品，然后解锁
- step 2：当条件不能满足时（仓库产品数量为 0），则调用 pthread_cond_wait 函数，这个函数具体做的事情如下:
    - 在线程睡眠之前，对互斥锁解锁
    - 让线程进入到睡眠状态
    - 等待条件变量收到信号时 **，该函数重新竞争锁，并获取锁后，函数返回 **
- step 3： 重新判断条件是否满足，如果不满足，则继续调用 pthread_cond_wait 函数
- step 4：唤醒后，从 pthread_cond_wait 返回，消费条件满足，则正常消费产品
- step 5：释放锁，整个过程结束
- 问题 1: 为什么条件变量需要与互斥锁结合起来使用?
- 解答:
    - 防止在调用 pthread_cond_wait 函数等待一个条件变量收到唤醒信号，另外一个线程发送信号在第一个线程实际等待它之前
    - 线程还没有完全进入到睡眠状态，其他线程发送唤醒信号
    - 下面是官方帮助文档的解释如下:

```
A  condition  variable must always be associated with a mutex, to avoid the race condition where a thread prepares to wait on a condition variable and another thread signals the condition just  before the first thread actually waits on it.
```

- 问题 2：在判断条件时，为什么需要使用 while(number == 0)，而不是 if()
- 解答:
    - 防止虚假唤醒
        - 能够唤醒的情况如下:
            - 被信号唤醒，并非由条件满足而唤醒
            - 条件变量状态改变时，一次唤醒多个线程，但是被其他线程先消费完产品，等到当前线程执行时，条件已经不满足



函数头文件 #include <pthread.h>

函数原型 int pthread_cond_wait(pthread_cond_t *restrict cond,

pthread_mutex_t *restrict mutex);

函数功能 阻塞线程，等待唤醒

函数参数

- cond：条件变量指针
- mutex：关联互斥锁指针

函数返回值

1. 注意
    1. 条件变量需要与互斥锁结合使用，先获得锁才能进行条件变量的操作
    2. 调用函数后会释放锁，并阻塞线程
    3. 一旦线程唤醒，需要重新竞争锁，重新获得锁之后，pthread_cond_wait 函数返回

```
These functions atomically release mutex and cause the calling thread to block on the condition variable cond;
```

3.3 pthread_cond_broadcast 与 pthread_condsingal

- pthread_cond_broadcast
  函数头文件 #include <pthread.h>
  函数原型 int pthread_cond_broadcast(pthread_cond_t *cond);
  函数功能 唤醒所有阻塞在某个条件变量上的线程
  函数参数 cond：条件变量指针
  函数返回值
  成功: 返回 0
  失败: 返回 错误码

- pthread_cond_signal

函数头文件 #include <pthread.h>

函数原型 int pthread_cond_signal(pthread_cond_t *cond);

函数功能 唤醒所有阻塞在某个条件变量上的线程

函数返回值

注意

- pthread_cond_signal 函数主要 适用等待线程都在执行完全相同的任务
- pthread_cond_broadcast 函数 主要适用等待线程都执行不相同的任务
- 条件变量并不保存状态信息，只是传递应用程序状态信息的一种通讯机制，发送信号时若无任何线程在等待该条件变量，则会被忽略
- 条件变量代表是一个通讯机制，用于传递通知与等待通知，用户可以设定条件来发送或者等待通知

示例

基于条件变量实现生产者与消费者模型（多个生产者对应一个消费者）

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


static int number = 0;

static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *thread_handler(void *arg)
{
    int cnt = atoi((char *)arg);
    int i,tmp;

    for(i = 0;i < cnt;i++){

        pthread_mutex_lock(&mtx);

        printf("线程 [%ld] 生产一个产品,产品数量为:%d\n",pthread_self(),++number);

        pthread_mutex_unlock(&mtx);
        pthread_cond_signal(&cond);
    }

    pthread_exit((void *)0);
}



int main(int argc,char *argv[])
{
    pthread_t tid;
    int i;
    int err;
    int total_of_produce = 0;
    int total_of_consume = 0;
    bool done = false;

    for (i = 1;i < argc;i++){
        total_of_produce += atoi(argv[i]);
        err = pthread_create(&tid,NULL,thread_handler,(void *)argv[i]);
        if (err != 0){
            perror("[ERROR] pthread_create(): ");
            exit(EXIT_FAILURE);
        }
    }


    for (;;){
        pthread_mutex_lock(&mtx);

        while(number == 0)
            pthread_cond_wait(&cond,&mtx);

        while(number > 0){
            total_of_consume++;
            printf("消费一个产品,产品数量为:%d\n",--number);
            done = total_of_consume >= total_of_produce;
        }

        pthread_mutex_unlock(&mtx);

        if (done)
            break;
    }
```

运行结果

程 [139950703310592] 生产一个产品, 产品数量为: 1

消费一个产品, 产品数量为: 0

线程 [139950703310592] 生产一个产品, 产品数量为: 1

消费一个产品, 产品数量为: 0

线程 [139950703310592] 生产一个产品, 产品数量为: 1

消费一个产品, 产品数量为: 0

线程 [139950720096000] 生产一个产品, 产品数量为: 1

消费一个产品, 产品数量为: 0

线程 [139950711703296] 生产一个产品, 产品数量为: 1

线程 [139950711703296] 生产一个产品, 产品数量为: 2

消费一个产品, 产品数量为: 1

消费一个产品, 产品数量为: 0

练习

实现多个生产者与多个消费者模型, 在示例的基础上进行修改, 提示，需要使用 pthread_cond_broadcast 函数唤醒所有阻塞的消费者线程

全文完