10.1 线程互斥锁_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 10.1 线程互斥锁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。

- 线程的主要优势在于，能够通过全局变量来共享信息, 不过这种便捷的共享是有代价的:
    - 必须确保多个线程不会同时修改同一变量
    - 某一线程不会读取正由其他线程修改的变量, 实际就是 不能让两个线程同时对临界区进行访问
    -
- 线程互斥锁则可以用于解决多线程资源竞争问题

示例

创建两个子线程，定义一个全局变量 global = 0, 子线程对这个全局变量进行加 1

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

static int global = 0;

void *do_thread(void *arg)
{
    int loops = *(int *)arg;
    int i,tmp = 0;

    for (i = 0;i < loops;i++){
        tmp = global;
        tmp++;
        global = tmp;
    }

    pthread_exit(NULL);
}

int main(int argc,char *argv[])
{
    int err,i = 0;
    pthread_t tid[2] = {0};
    int loops = 0;

    if (argc != 2){
        fprintf(stderr,"Usage : < %s > < count loops>\n",argv[0]);
        exit(EXIT_FAILURE);
    }

    loops = atoi(argv[1]);

    for (i = 0;i < 2;i++){
        err = pthread_create(&tid[i],NULL,do_thread,&loops);
        if (err != 0){
            fprintf(stderr,"[ERROR] pthread_create(): < %s > \n",strerror(err));
            exit(EXIT_FAILURE);
        }
    }


    pthread_join(tid[0],NULL);
    pthread_join(tid[1],NULL);

    printf("global = %d\n",global);
    return 0;
}
```

运行结果

./a.out 100

global = 200

./a.out 10000000

global = 10833956

- 线程互斥锁工作机制:
  - 当一个线程 A 获得锁, 另外一个线程 B 在获得锁则会阻塞，直到线程 A 释放锁, 线程 B 则才会获得锁.
  -
- 线程互斥锁工作原理:
  - 本质上是一个 pthread_mutex_t 类型的变量, 假设名为 v
    - 当 v = 1, 则表示当前临界资源可以竞争访问 , 得到互斥锁的线程则可以访问，此时 v = 0
    - 当 v = 0, 则表示临界资源正在被某个线程访问, 其他线程则需要等待
- 线程互斥锁的特点
  - 互斥锁是一个 pthread_mutex_t 型的变量, 就代表一个 互斥锁
  - 如果两个线程访问的是同一个 pthread_mutex_t 变量，那么它们访问了同一个互斥锁
  - 对应的变量定义在 pthreadtypes.h 头文件中, 是一个共用体中包含一个结构体
  -
```
67 typedef union
68 {
69   struct __pthread_mutex_s __data;
70   char __size[__SIZEOF_PTHREAD_MUTEX_T];
71   long int __align;
72 } pthread_mutex_t;
```
- 线程互斥锁的初始化方式主要分为两种
  - 静态初始化
    - 定义 pthread_mutex_t 类型的变量，然后对其初始化为 PTHREAD_MUTEX_INITIALIZER.
    -
      1. `pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER`
  - 动态初始化
    - 动态初始化主要涉及两个函数 pthread_mutex_init 函数 与 pthread_mutex_destroy 函数
      -
      - pthread_mutex_init 函数
        - 函数头文件 #include <pthread.h>
        - 函数原型 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
        - const pthread_mutexattr_t *restrict attr);
        - 函数功能 初始化线程互斥锁
        - 函数参数
        -
          - mutex：线程互斥锁对象指针
        -
          - attr：线程互斥锁属性
        - 函数返回值
        -
          - 成功：返回 0
        -
          - 失败：返回 错误编码
        -

- pthread_mutex_destroy 函数
  - 函数头文件 #include <pthread.h>
  - 函数原型 int pthread_mutex_destroy(pthread_mutex_t *mutex);
  - 函数功能 销毁线程互斥锁
  - 函数参数 mutex：线程互斥锁指针
  - 函数返回值
    - 成功：返回 0
    - 失败：返回 错误编码

- 线程互斥锁的操作主要分为 获取锁 (lock) 与 释放锁 (unlock)，具体函数描述如下
  - pthread_mutex_lock
    - 函数头文件 #include <pthread.h>
    - 函数原型 int pthread_mutex_lock(pthread_mutex_t *mutex);
    - 函数功能 将互斥锁进行锁定，如果已经锁定，则阻塞线程
    - 函数参数 mutex：线程互斥锁指针
    - 函数返回值
      - 成功：返回 0
      - 失败：返回 错误码
  - pthread_mutex_unlock
    - 函数头文件 #include <pthread.h>
    - 函数原型 int pthread_mutex_unlock(pthread_mutex_t *mutex);
    - 函数功能 解除互斥锁锁定状态，解除后，所有线程可以重新竞争锁
    - 函数参数 mutex：线程互斥锁对象的指针
    - 函数返回值
      - 成功：返回 0
      - 失败：返回 错误码

示例

创建两个线程，分别对全局变量进行 +1 操作, 分别对比使用互斥锁和没有使用互斥锁

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

static int global = 0;
```

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *do_thread(void *arg)
{
    int loops = *(int *)arg;
    int i,tmp = 0;

    for (i = 0;i < loops;i++){
        pthread_mutex_lock(&mutex);
        tmp = global;
        tmp++;
        global = tmp;
        pthread_mutex_unlock(&mutex);
    }

    pthread_exit(NULL);
}

int main(int argc,char *argv[])
{
    int err,i = 0;
    pthread_t tid[2] = {0};
    int loops = 0;

    if (argc != 2){
        fprintf(stderr,"Usage : < %s > < count loops>\n",argv[0]);
        exit(EXIT_FAILURE);
    }

    loops = atoi(argv[1]);

    for (i = 0;i < 2;i++){
        err = pthread_create(&tid[i],NULL,do_thread,&loops);
        if (err != 0){
            fprintf(stderr,"[ERROR] pthread_create(): < %s > \n",strerror(err));
            exit(EXIT_FAILURE);
        }
    }


    pthread_join(tid[0],NULL);
    pthread_join(tid[1],NULL);

    printf("global = %d\n",global);
    return 0;
}
```

- 没有使用互斥锁时，输出结果如下:

./a.out 1000000

global = 1321859

global = 2000000

全文完