一、前言
二、Peterson 算法简介
三、测试代码
四、Mutex 互斥锁对代码执行效率的影响
1. 单线程中：Mutex 互斥锁对代码执行效率的影响
2. 多线程中：Mutex 互斥锁对代码执行效率的影响
3. 在两个线程中，使用 Peterson 算法来保护临界区
五、总结

一、前言

在 Linux 系统中，当多个线程并行执行时，如果需要访问同一个资源，那么在访问资源的地方，需要使用操作系统为我们提供的同步原语来进行保护。同步原语包括：互斥锁、条件变量、信号量等，被保护的代码称作“临界区”。

这是非常正规的流程，我们基本上也都是这么做的。

那有没有想过，这些同步原语对代码的执行效率会产生多大的影响？是否可以不使用操作系统提供的这些机制，而是用其它纯软件的方法也能达到保护临界区的目的呢？

这篇文章我们介绍一下 Peterson(皮特森)算法，也许实用性不强，但是可以给我们带来一些思考，提高我们的编程元技能。

二、Peterson 算法简介

这个算法主要用来解决临界区的保护问题。我们知道，一个临界区必须保证 3 个条件：

1. 互斥访问: 在任意一个时刻，最多只能有一个线程可以进入临界区；
2. 空闲让进：当没有线程正在执行临界区的代码时，必须在所有申请进入临界区的线程中，选择其中的一个，让它进入临界区；
3. 有限等待：当一个线程申请进去临界区时，不能无限的等待，必须在有限的时间内获得许可进入临界区。也就是说，不论其优先级多低，不应该饿死在该临界区入口处。

Peterson算法是一个实现互斥锁的并发程序设计算法，可以控制两个线程访问一个共享的用户资源而不发生访问冲突。

Peterson 算法是基于双线程互斥访问的 LockOne 与 LockTwo 算法而来。

1. LockOne 算法使用一个 flag 布尔数组
2. LockTwo 使用一个 turn 的整型量

这 2 个算法都实现了互斥，但是都存在死锁的可能。Peterson 算法把这两种算法结合起来，完美地用软件实现了双线程互斥问题。

算法说明如下

两个重要的全局变量：

flag 数组：有 2 个布尔元素，分别代表一个线程是否申请进入临界区；
turn：如果 2 个线程都申请进入临界区，这个变量将会决定让哪一个线程进入临界区；

三、测试代码

```
// 被 2 个线程同时访问的全局资源
static int num = 0;
```

```
BOOL flag[2] = { 0 };
int turn = 0;

void *thread0_routine(void *arg)
{
    for (int i = 0; i < 1000000; ++i)
    {
        flag[0] = TRUE;
        turn = 1;
        while (TRUE == flag[1] && 1 == turn);

        // 临阶区代码
        num++;

        flag[0] = FALSE;
    }

    return NULL;
}

void *thread1_routine(void *arg)
{
    for (int i = 0; i < 1000000; ++i)
    {
        flag[1] = TRUE;
        turn = 0;
        while (TRUE == flag[0] && 0 == turn);

        // 临阶区代码
        num++;

        flag[1] = FALSE;
    }

    return NULL;
}
```

全局资源 num 的初始值为 0 ，两个编程分别递增 100 万次，因此最终结果应该是 200 万，实际测试结果也确实如此。

四、Mutex 互斥锁对代码执行效率的影响

1. 单线程中：Mutex 互斥锁对代码执行效率的影响

```
for (int i = 0; i < 1000000; ++i)
{
    num++;
}
```

以上代码，耗时约：1.8ms -- 3.5ms。

```
for (int i = 0; i < 1000000; ++i)
{
    pthread_mutex_lock(&mutex);
    num++;
    pthread_mutex_unlock(&mutex);
}
```

以上代码，耗时约：23.9ms -- 38.9ms。可以看出，上锁和解锁对代码执行效率的影响还是很明显的。

2. 多线程中：Mutex 互斥锁对代码执行效率的影响

```
void *thread0_routine(void *arg)
{
    for (int i = 0; i < 1000000; ++i)
    {
        pthread_mutex_lock(&mutex);
        num++;
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}

void *thread1_routine(void *arg)
{
    for (int i = 0; i < 1000000; ++i)
    {
        pthread_mutex_lock(&mutex);
        num++;
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}
```

耗时：

thread0: diff = 125.8ms
thread1: diff = 129.1ms

3. 在两个线程中，使用 Peterson 算法来保护临界区

耗时：

thread1: diff = 1.89ms
thread0: diff = 1.94ms

五、总结

Peterson 算法使用纯软件来保护临界区，比使用操作系统提供的互斥锁表现出了更好的性能。

但是它也有一个缺点：只能使用在 2 个线程中，但是由于它与平台无关，在某些特殊的场合，也许能够拿来为我们所用！

推荐阅读

【C 语言】

1. C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
2. 原来gdb的底层调试原理这么简单
3. 一步步分析-如何用C实现面向对象编程
4. 提高代码逼格的利器：宏定义-从入门到放弃
5. 利用C语言中的setjmp和longjmp，来实现异常捕获和协程

【应用程序设计】

1. 都说软件架构要分层、分模块，具体应该怎么做(一)
2. 都说软件架构要分层、分模块，具体应该怎么做(二)
3. 物联网网关开发：基于MQTT消息总线的设计过程(上)
4. 物联网网关开发：基于MQTT消息总线的设计过程(下)
5. 我最喜欢的进程之间通信方式-消息总线

【操作系统】

1. 为什么航天器、导弹喜欢用单片机，而不是嵌入式系统?

【物联网】

1. 关于加密、证书的那些事
2. 深入LUA脚本语言，让你彻底明白调试原理

【胡说八道】

1. 以我失败的职业经历：给初入职场的技术人员几个小建议