一、前言

在上一篇文章中，介绍了一种纯软件算法，用来实现临界区的保护功能，文章链接: C语言边角料2：用纯软件来代替Mutex互斥锁。

首先明确一下：如果利用操作系统提供的互斥锁可以实现我需要的功能，我肯定使用互斥锁，之所以介绍 Peterson 这个算法，主要是因为它比较有意思，很小巧，可以为我们带来一些"规范的"编程之外的一些想法。

后台也有一些小伙伴对这个算法发表了一些留言，只要有想法都非常好，就怕不去想。

其中有位朋友提到，这个算法只能用在 2 个线程中，是否有其他的类似算法，可以用在多线程中?

晚上下班后，我就花了点时间找到下面的这个算法，分享一下!

二、Micha Hofri 算法

这个算法我没有找到名字，暂且以作者的名字来称呼这个算法吧!

算法截图:

---

0. global integer arrays $q[n]$, $turn[n-1]$
1. p_i: repeat forever
2. { non-critical part of p_i
3. for j=1 to $n-1$ do
4. { $q[i] = j$
5. $turn[j] = i$
6. L: for k=1 to $i-1$, $i+1$ to n
7. if (($q[k] \geq j$) and ($turn[j] = i$)) goto L
8. }
9. $q[i] = n$; Critical section of p_i (CS_i)
10. $q[i] = 0$
11. }

Lines 3–8 are the entry protocol; line 10 is the exit protocol. An equivalent formulation of lines 6–7 could be used:

(6-7)′ wait until ((for all $k \neq i$ $q[k] < j$) or ($turn[j] \neq i$)).

从算法的主体代码看，Hofri 算法主要是扩展了 Peterson 算法，都是使用 2 个全局变量数组来控制哪个线程可以进入临界区。

这个算法的论证比较复杂，都是一些数学方面的证明，文章在这里：Proof of a Mutual Exclusion Algorithm-- A `Class'ic Example， 1989 年发表，感兴趣的小伙伴可以自行去烧脑研究。

三、测试代码

```
// 线程操作的资源
static int num = 0;

// 创建 10 个线程
#define THREAD_NUM      10

// 这 2 个全局变量控制算法
int flag[THREAD_NUM] = {0 };
int turn[THREAD_NUM - 1] = {0};

// 这是线程函数
void *thread_routine(void *arg)
{
    int index = *(int *)arg;

    for (int i = 0; i < 10000; ++i)
    {
        for (int j = 1; j < THREAD_NUM - 1; j++)
        {
            flag[index] = j;
            turn[j] = index;
    L:
            for (int k = 1; k < THREAD_NUM; ++k)
            {
                if (k == index) continue;
                if ((flag[k] >= j) && turn[j] == index)
                    goto L;
            }

        }

        flag[index] = THREAD_NUM;

        // 关键代码段
        num++;

        flag[index] = 0;
    }
    return NULL;
}

void test()
{
    // 用来传递线程的索引
    int index[THREAD_NUM] = {0};

    创建多个线程，执行同一个函数
    pthread_t t[THREAD_NUM];
    for (int i = 0; i < THREAD_NUM; ++i)
    {
        index[i] = i;
        pthread_create(&t[i], NULL, thread_routine, &index[i]);
    }
}
```

编译、执行，所有线程执行结束后，共享资源 num 变量可以得到正确的结果。

四、总结

还是重复一下文章开头说的话，这里的算法仅仅是说明它可以完成保护临界区的功能，但是在实际项目中，真心不建议这么来用，毕竟代码的可维护性是非常重要的！

好文章，要转发；越分享，越幸运！

星标公众号，能更快找到我！

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