操作系统

陈彦帆 2018K8009918002

（1）Peterson 算法

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#define M 10000000
int data[M];
int idx;
volatile int flag[2];
volatile int turn;

static inline unsigned long rdtsc(void)
{
    unsigned long tickl,tickh;
    asm volatile   (
        "rdtsc\n\t"
        :"=a"(tickl),"=d"(tickh));
    return ((unsigned long )tickh<<32)|tickl;
}

#define noreorder() \
    asm volatile( \
        "mfence\n\t" \
        : \
    )
void *thread_1(void *arg)
{
    for(int i=0;i<M;){
        flag[1] = 1;
        noreorder();
        turn = 0;
        noreorder();
        while(flag[0] && turn == 0);
        for(int j=0;j<100;j++){
            data[idx] = i + 1;
            idx++;
            i+=2;
        }
        flag[1] = 0;
    }
    return NULL;
}

void *thread_0(void *arg)
{
    for(int i=0;i<M;){
        flag[0] = 1;
        noreorder();
        turn = 1;
        noreorder();
        while(flag[1] && turn == 1);
        for(int j=0;j<100;j++){
            data[idx] = i;
            idx++;
            i+=2;
        }
        flag[0] = 0;
```

```c
    }
    return NULL;
}

int main(void)
{
    unsigned long tick1,tick2;
    pthread_t thread0, thread1;
    tick1 = rdtsc();
    pthread_create(&thread0, NULL, thread_0, NULL);
    pthread_create(&thread1, NULL, thread_1, NULL);
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);

    tick2 = rdtsc();

    int maxDiff = 0;
    int maxidx = 0;
    #define abs(x) ((x)<0?-(x):(x))
    for(int i=0;i<M-1;i++){
        int diff = abs(data[i] - data[i+1]);
        if(diff>maxDiff){
            maxDiff = diff;
        }
    }

    printf("Index = %d\nMax diff is: %d\nTime is %lu clocks\n", idx,maxDiff,tick2-tick1);
    return 0;
}
```

注意以下代码段：

```c
     flag[1] = 1;
    noreorder();
    turn = 0;
    noreorder();
           while(flag[0] && turn == 0);
```

当没有加入中间的 noreorder( )时，即为课本上的标准写法。但在实际运行过程中可能出现问题：

```
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
Max diff is: 4397
Time is 265150338 clocks
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 9999913
Max diff is: 9999999
Time is 344904425 clocks
```

有时候运行结果符合预期（index 刚好加到一千万），有时候 index 结果不到一千万。

首先怀疑编译器调整了执行顺序。通过 objdump 命令查看反汇编文件，发现编译器并没有调节语句的前后顺序：

```
7a7: c7 05 b3 62 82 02 01      movl     $0x1,0x28262b3(%rip)            # 2826a64 <flag+0x4>
 7ae:      00 00 00
 7b1:      c7 05 85 08 20 00 00      movl     $0x0,0x200885(%rip)            # 201040 <turn>
 7b8:      00 00 00
 7bb:      90                                   nop
```

```
7bc:     8b 05 9e 62 82 02        mov      0x282629e(%rip),%eax           # 2826a60 <flag>
7c2:     85 c0                         test     %eax,%eax
7c4:     74 0a                         je       7d0 <thread_1+0x3a>
7c6:     8b 05 74 08 20 00        mov      0x200874(%rip),%eax          # 201040 <turn>
7cc:     85 c0                         test     %eax,%eax
7ce:     74 ec                         je       7bc <thread_1+0x26>
```

然后怀疑 cpu 乱序执行。flag 的赋值和 turn 的赋值，turn 的赋值和 while 语句中 flag 判断是可能被乱序执行的。为此，在两条语句中间插入 x86 的 mfence 指令，表示在 mfence 指令前的读写操作当必须在 mfence 指令后的读写操作前完成。这样就避免被乱序执行出错。（也可在语句间填充大量的 nop 指令）

经过 100 次以上测试，运行结果符合预期。

```
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
Max diff is: 16597
Time is 251508201 clocks
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
Max diff is: 20597
Time is 242995877 clocks
```

（2）使用互斥锁

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

#define M 10000000
int data[M];
int idx;
pthread_mutex_t the_mutex;

static inline unsigned long rdtsc(void)
{
    unsigned long tickl,tickh;
    asm volatile   (
        "rdtsc\n\t"
        :"=a"(tickl),"=d"(tickh));
    return ((unsigned long )tickh<<32)|tickl;
}


void *thread_1(void *arg)
{
    for(int i=0;i<M;){
        pthread_mutex_lock(&the_mutex);
        for(int j=0;j<100;j++){
            data[idx] = i + 1;
            idx++;
            i+=2;
        }
        pthread_mutex_unlock(&the_mutex);
    }
    return NULL;
}
```

```
void *thread_0(void *arg)
{
    for(int i=0;i<M;){
        pthread_mutex_lock(&the_mutex);
        for(int j=0;j<100;j++){
            data[idx] = i;
            idx++;
            i+=2;
        }
        pthread_mutex_unlock(&the_mutex);
    }
    return NULL;
}

int main(void)
{
    unsigned long tick1,tick2;
    pthread_t thread0, thread1;
    tick1 = rdtsc();
    pthread_create(&thread0, NULL, thread_0, NULL);
    pthread_create(&thread1, NULL, thread_1, NULL);
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);
    tick2 = rdtsc();
    int maxDiff = 0;
    #define abs(x) ((x)<0?-(x):(x))
    for(int i=0;i<M-1;i++){
        int diff = abs(data[i] - data[i+1]);
        if(diff>maxDiff)
            maxDiff = diff;
    }

    printf("Index = %d\nMax diff is: %d\nTime is %lu clocks\n", idx,maxDiff,tick2-tick1);
    return 0;
}
```

运行结果符合预期：

```
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
Max diff is: 2424997
Time is 124594055 clocks
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
Max diff is: 4009797
Time is 118677620 clocks
```

（3）使用__sync_fetch_and_add 函数

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

#define M 10000000
int data[M];
int idx;

static inline unsigned long rdtsc(void)
{
    unsigned long tickl,tickh;
    asm volatile   (
        "rdtsc\n\t"
        :"=a"(tickl),"=d"(tickh));
    return ((unsigned long )tickh<<32)|tickl;
}


void *thread_1(void *arg)
{
    for(int i=0;i<M;){

        for(int j=0;j<100;j++){
            int temp = __sync_fetch_and_add(&idx,1);
            data[temp] = i + 1;
            i+=2;
        }

    }
    return NULL;
}

void *thread_0(void *arg)
{
    for(int i=0;i<M;){
        for(int j=0;j<100;j++){
            int temp = __sync_fetch_and_add(&idx,1);
            data[temp] = i;
            i+=2;
        }
    }
    return NULL;
}

int main(void)
{
    unsigned long tick1,tick2;
    pthread_t thread0, thread1;
    tick1 = rdtsc();
    pthread_create(&thread0, NULL, thread_0, NULL);
    pthread_create(&thread1, NULL, thread_1, NULL);
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);

    tick2 = rdtsc();

    int maxDiff = 0;
    #define abs(x) ((x)<0?-(x):(x))
    for(int i=0;i<M-1;i++){
        int diff = abs(data[i] - data[i+1]);
        if(diff>maxDiff)
            maxDiff = diff;
    }

    printf("Index = %d\nMax diff is: %d\nTime is %lu clocks\n", idx,maxDiff,tick2-tick1);
    return 0;
}
```

运行结果符合预期:

```
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
```

```
Max diff is: 284279
Time is 583407554 clocks
alphabet@ubuntu:~/cpro/test$ ./t1
Index = 10000000
Max diff is: 392623
Time is 602387005 clocks
```

（4）比较三种方式的执行时间
具体执行时间见上文。经计算得
时间比例 Peterson 算法：互斥锁：原子取加 = 2.03：1：4.88

猜测：互斥锁比 peterson 算法快是因为它通过硬件的原子指令完成，而 peterson 算法包含忙等待。原子取加最慢是因为每一次加法都要进行一个原子操作，而前两种方法是每 100 次加法才进入临界区一次。

测试机器为 Intel i7-8550u VMware15 Ubuntu18.04 64bit