Q:CAS的实现

A:gcc提供了两个函数

bool \_\_sync\_bool\_compare\_and\_swap (type \*ptr, type oldval, type newval, ...)  
type \_\_sync\_val\_compare\_and\_swap (type \*ptr, type oldval, type newval, ...)  
这两个函数提供原子的比较和交换，如果\*ptr == oldval,就将newval写入\*ptr,  
第一个函数在相等并写入的情况下返回true，这个函数比第二个好在，返回bool值可以知道有没有更新成功.  
第二个函数在返回操作之前的值。

第二个函数用c语言描述：

type \_\_sync\_val\_compare\_and\_swap (type \*ptr, type oldval, type newval, ...)

{

type cur = \*ptr;

if (cur == oldval)

{

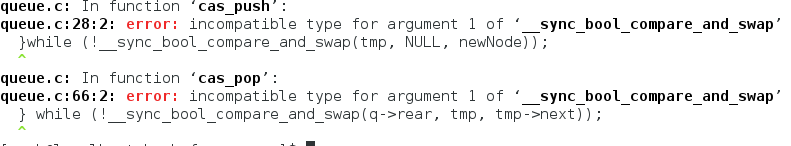
\*ptr = newval;

}

return cur;// 返回操作之前的值

}

type只能是1,2,4或8字节长度的int类型，否则会发生下面的错误



Q: 操作系统级别是如何实现的

A: X86中有一个CMPXCHG的汇编指令

Q: CAS指令有什么缺点

A:

1.存在ABA问题因为CAS需要在操作值的时候检查下值有没有发生变化，如果没有发生变化则更新，但是如果一个值原来是A，变成了B，又变成了A，那么使用CAS进行检查时会发现它的值没有发生变化，但是实际上却变化了。ABA问题的解决思路就是使用版本号。在变量前面追加上版本号，每次变量更新的时候把版本号加一，那么A－B－A 就会变成1A-2B－3A。

2.循环时间长开销大自旋CAS如果长时间不成功，会给CPU带来非常大的执行开销。

3. 只能保证一个共享变量的原子操作对一个共享变量执行操作时，我们可以使用循环CAS的方式来保证原子操作，但是对多个共享变量操作时，循环CAS就无法保证操作的原子性，这个时候就可以用锁，或者有一个取巧的办法，就是把多个共享变量合并成一个共享变量来操作。比如有两个共享变量i＝2,j=a，合并一下ij=2a，然后用CAS来操作ij。

gcc从4.1.2提供了\_\_sync\_\*系列的built-in函数，用于提供加减和逻辑运算的原子操作。  
  
其声明如下：

原子操作的后置加加type \_\_sync\_fetch\_and\_add (type \*ptr, type value, ...)

原子操作的前置加加type \_\_sync\_add\_and\_fetch (type \*ptr, type value, ...)  
其他类比

type \_\_sync\_fetch\_and\_sub (type \*ptr, type value, ...)  
type \_\_sync\_fetch\_and\_or (type \*ptr, type value, ...)  
type \_\_sync\_fetch\_and\_and (type \*ptr, type value, ...)  
type \_\_sync\_fetch\_and\_xor (type \*ptr, type value, ...)  
type \_\_sync\_fetch\_and\_nand (type \*ptr, type value, ...)  
  
  
type \_\_sync\_sub\_and\_fetch (type \*ptr, type value, ...)  
type \_\_sync\_or\_and\_fetch (type \*ptr, type value, ...)  
type \_\_sync\_and\_and\_fetch (type \*ptr, type value, ...)  
type \_\_sync\_xor\_and\_fetch (type \*ptr, type value, ...)  
type \_\_sync\_nand\_and\_fetch (type \*ptr, type value, ...)  
这两组函数的区别在于第一组返回更新前的值，第二组返回更新后的值。

关于CAS函数，参考: <http://blog.csdn.net/youfuchen/article/details/23179799>