基于arm平台独占访问指令实现互斥锁和信号量

一、ldrex和strex指令

为实现多核的数据安全共享，需要在处理共享数据代码之前加互斥锁(mutex)或者信号量(semaphore)，并且在执行完后释放互斥锁或信号量，为了实现互斥锁和信号量，一般需要平台提供支持。ldrex和strex指令就是arm平台下提供的制作互斥锁和信号量的基础。ldrex和strex指令将更新内存的原子操作分成了两个独立的步骤。这两个独立的步骤和独占监视器一起使得内存更新为原子性。

ldrex

ldrex指令用来读取内存中的值，并且在监视器上标记对该段内存的独占访问:

```
ldrex r1, [r0]
```

比如以上指令意味着，读取r0指向的4字节内存的值，将其保存到r1，同时在监视器中标记对r0指向的内存区域的独占访问。如果执行ldrex指令的时候发现该段内存已被标记为独占访问了，并不影响该指令的执行。

strex

strex指令在更新内存数据时，通过检查该段内存是否被标记为独占访问，并以此来决定是否更新该内存的数据：

```
strex r1, r2, [r3]
```

比如执行以上指令的时候发现r3指向的内存已经被标记为独占访问了，则将r2中的值更新到r3指向的内存，并将r1设置为0，表示指令执行成功，最后将独占访问标记清除。如果执行以上指令的时候发现r3指向的内存没有被标记为独占访问，则不会将r2中的值更新到r3指向的内存中，并将r1设置为1，表示指令执行失败。因此当某条strex指令执行成功后，以后再对同一段内存尝试使用strex指令更新时都会失败，因为独占访问标记已经被清空。因此**核心思想就是无论有多少处理器，多少个地方申请对同一段内存操作，保证只有最早的更新可以成功，之后的更新都会失败，失败了就证明对该段内存访问有冲突。实际应用中，在strex失败后通过重新ldrex读取该段内存的数据，再处理一次，再尝试strex保存，直到成功为止。**

二、Exclusive monitors

仅靠cpu是无法实现ldrex和strex指令的，它需要独占监视器的支持。独占监视器是一个只有开放(open)和独占(exclusive)两种状态的简单状态机，其结构如下：

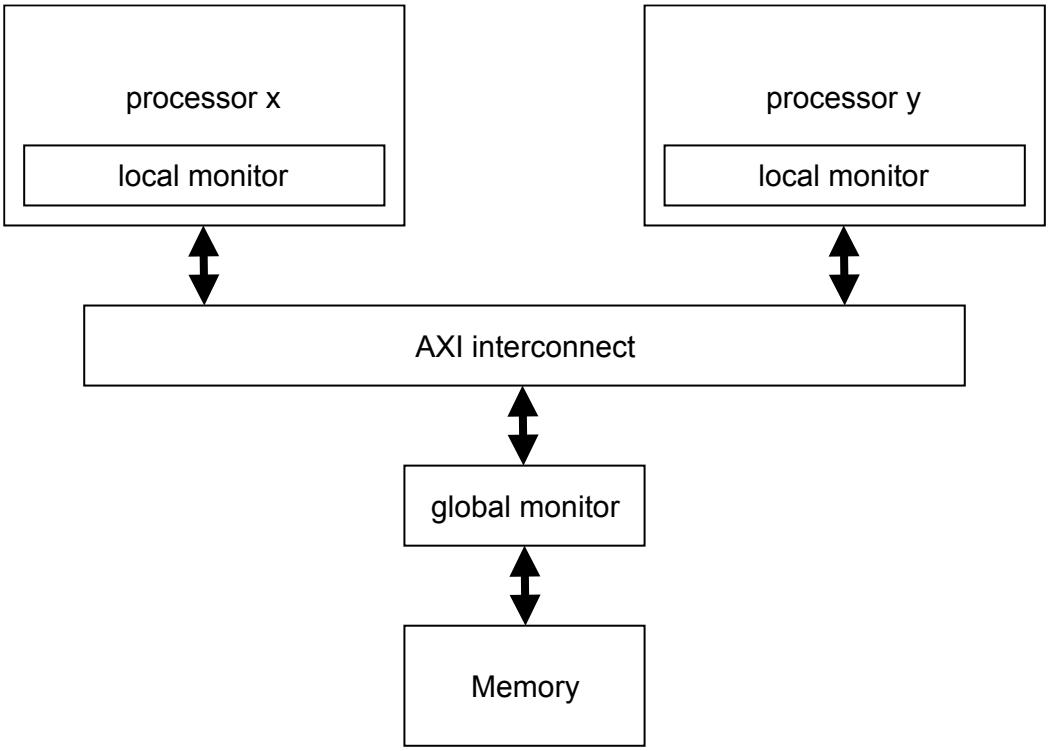

图 多核系统下的全局监视器和本地监视器

如上所示，有本地监视器和全局监视器两种。每个处理器内部都有一个本地监视器(local monitor)，且在整个系统范围内还有一个全局监视器(global monitor)。

如果对非共享内存区域进行独占访问，只需要设置和观察处理器内部的本地监视器；如果要对共享内存区域进行独占访问，需要设置和观察处理器内部的本地监视器和处理器共享的全局监视器。

本地监视器只会标记本处理器对某段内存的独占访问；全局监视器会标记对应处理器对某段内存的独占访问，当另外一个处理器通过ldrex申请对同一共享内存段进行访问时，会设置这另外的处理器对同一段内存的独占访问，此时全局监视器就有两个处理器对同一内存段的两个标记。

对于全局监视器来说，当发生以下两种情况时，会清除某个处理器对某段内存的独占访问标记：

1）当该处理器调用ldrex指令申请独占访问另一段内存时，在清除之前该处理器对之前的内存访问标记的同时，设置该处理器对新申请的独占访问内存的独占标记。

2）当别的处理器通过strex成功更新了该段独占访问内存时，也会同时将本处理器对该内存的独占标记清除掉。

对于本地监视器来说，有可能实现了和全局监视器一样的地址标记功能，也有可能只实现了ldrex和strex的开放状态和独占状态跟踪，意思就是无论对于哪段内存使用ldrex，只会简单将本地监视器设置为独占状态，不会标记地址，下一次无论对哪段内存使用strex都会将本地监视器设置回开放状态。为了保险起见，我们不要认为实现了地址标记功能。因此当发生以下情况时，会清除本地监视器的独占状态：

1）当本地监视器的处理器使用strex更新任何一段内存时。

三、利用ldrex和strex实现互斥锁和信号量

```
/* mutex and semaphore */
void lock_mutex(void *mutex);
void unlock_mutex(void *mutex);
void sem_dec(void *semaphore);
void sem_inc(void *semaphore);
```

首先给出上面代码的互斥锁和信号量的接口使用说明，mutex是锁的地址；lock_mutex是上锁操作，如果当前锁被占用，则阻塞直到锁在其他地方被释放，然后才能上锁成功；unlock_mutex是解锁操作，处理完加锁区域的代码后释放锁；semaphore是信号量地址；sem_dec是使用了一个信号量操作，当调用sem_dec时如果信号量大于0则调用成功，否则阻塞直到别的地方释放信号量；sem_inc是释放了一个信号量操作，使得信号量加1。

```
@@ for mutex and semaphore
.equ    locked,     0x0
.equ    unlocked,   0x1

.global lock_mutex
.align 4
lock_mutex:
    ldr     r1, =locked
1:
    ldrex   r2, [r0]
    cmp     r2, r1
    beq     2f              @ if locked, wait for it to be released from 2
    strexne r2, r1, [r0]    @ not locked, attempt to lock it
    cmpne   r2, #1          @ check if store-exclusive failed
    beq     1b              @ failed, retry immediately
    dmb                     @ successed, required before accessing protected resource
    bx      lr              @ going to access protected resource
2:
    wfe                     @ wait the lock owner to notify that it has released the mutex
    b       1b              @ wake and retry


.global unlock_mutex
.align 4
unlock_mutex:
    ldr     r1, =unlocked
    dmb                     @ required before releasing protected resource
    str     r1, [r0]        @ unlock mutex
    dsb
    sev                     @ notify other lock waiters that the mutex has been unlocked
    bx      lr              @ complete accessing protected resource



.global sem_dec
.align 4
sem_dec:
1:
    ldrex   r1, [r0]
    cmp     r1, #0          @ test if the semaphore holds the value 0
    beq     2f              @ if it does, block before retrying
    sub     r1, r1, #1      @ if not, decrement temporary copy
    strex   r2, r1, [r0]    @ attempt store-exclusive
    cmp     r2, #1          @ check if store-exclusive failed
    beq     1b              @ failed, retry immediately
    dmb                     @ successed, required before accessing protected resource
    bx      lr              @ going to access protected resource
2:
```

```
    wfe                     @ wait other semaphore owners to notify that it has released
semaphore
    b       1b              @ wake and retry


.global sem_inc
.align 4
sem_inc:
1:
    ldrex   r1, [r0]
    add     r1, r1, #1      @ increment temporary copy
    strex   r2, r1, [r0]    @ attempt store-exclusive
    cmp     r2, #1          @ check if store-exclusive failed
    beq     1b              @ failed, retry immediately
    dsb                     @ successed
    sev                     @ notify other waiters that the semaphore has been decremented
by 1
    bx      lr              @ complete accessing protected resource
```

以上是该四个用户使用接口的具体实现。

lock_mutex首先通过ldrex指令加载r0指向的内存数据，即传进来的接口参数mutex对应的数据，检测它是否锁住，如果以锁住则跳到标号2处通过wfe进入低功耗的阻塞状态，如果没有锁住，则尝试通过strex更新mutex处的数据为锁定状态，如果strex失败则从新回到1：处重新申请独占访问，如果strex成功则返回去处理加锁区域的代码。

unlock_mutex直接使用str来更新mutex处的值来释放锁，因为只有当前例程拥有该锁，在释放锁后通过sev指令来唤醒别处可能通过wfe阻塞的例程，最后再返回。

sem_dec实现与lock_mutex几乎一样，除了信号量是可以大于1，而互斥锁只能为0或1之外没有什么需要注意的地方，这里不再赘述。

而sem_inc实现与unlock_mutex有一个特别要注意的差别，就是释放信号量需要独占访问操作来更新，因为可能不止一个例程拥有信号量。