The Mesa system forced the developers to address a number of the practical issues associated with monitors and condition variables.  Describe some aspects of the implementation of synchronization primitives in Mesa that were not addressed by C.A.R. Hoare.

*The aspects of the implementation of synchronization primitives in Mesa that were not addressed by C.A.R. Hoare:*

*the semantics of nested monitor calls;*

*the various ways of defining the meaning of WAIT;*

*priority scheduling;*

*handling of timeouts, aborts and other exceptional conditions;*

*interactions with process creation and destruction;*

*monitoring large numbers of small objects.*

*Monitor vs semaphore vs mutex:*

*A monitor is an object that only allow one thread to execute it provided procedures. It is kinda like a simple OS. It can be implemented with a semaphore. A semaphore is an OS implemented low level object that has P and V operations to allow a fix number of processes access a shared resource.*

*A mutex is an OS implemented low level object and only allow one process to access a shared resource at a time. Mutex can implement monitor and monitor can implement mutex.*

* P/V操作必须成对出现，这样就不会出现死循环
* 信号量大于0的确表示有临界资源可供使用，而且这个时候没有进程被阻塞在这个资源上，也就是说没有进程因为得不到这类资源而阻塞，所以没有被阻塞的进程，自然不需要唤醒。
* 原语操作的本质在于：一个进程使用完临界资源后，释放临界资源，使信号量加1，以通知其它的进程，这个时候如果信号量<=0，表明有进程阻塞在该类资源上，因此要从阻塞队列里唤醒一个进程来“转手”该类资源。比如，有2个某类资源，三个进程A、B、C、D要用该类资源，最开始信号量=2，当A进入，信号量=1，当B进入信号量=0，表明该类资源刚好用完，当C进入时信号量=-1，表明有一个进程被阻塞了（C会被阻塞），D进入，信号量=-2（D继续阻塞）。当A用完该类资源时，进行V操作，信号量=-1，释放该类资源，而这时信号量<0,表明有进程阻塞在该类资源上，于是唤醒一个.
* 当一个进程阻塞了的时候，它已经执行过了P操作，并卡在临界区那个地方。当唤醒它时就立即进入它自己的临界区，并不需要执行P操作了，当执行完了临界区的程序后，就执行V操作。
* 当信号量信号量小于0时，其绝对值表示系统中因请求该类资源而被阻塞的进程数目。S大于0时表示可用的临界资源数。注意在不同情况下所表达的含义不一样。当等于0时，表示刚好用完。

信号量已经提供了一个方便且高效的进程同步机制，但是信号量有个缺点就是每次都需要程序员专门的去调用PV操作，如果程序员由于大意调用错了PV操作，比如该调用P操作的时候却调用了V操作，该针对X信号量调用P操作，却对Y信号量调用了P操作。这种错误是非常危险的，因为进程同步的问题不是每次都能重现，比如前面的错误在测试环境可能不会出现错误，但是到了生产环境就有可能出现，且无法重现。

为了解决上述问题，引入了管程（Monitor），信号量是操作系统层面的结构，管程是一个程序结构，由编程语言封装，最终由编译器：

* 一个管程定义了一个数据结构和能够并发进程所执行的一组操作，这组操作能同步进程和改变管程中的数据
* 进程可以调用管程的操作，但是不能访问管程的内部数据结构
* 它表示一个对象自己维护自己的状态，并且能够根据自身状态来同步并发的线程操作，而不是把这种同步的手段交给调用者来处理。
* 管程保证同一时间只有一个进程处在管程内部的方法内