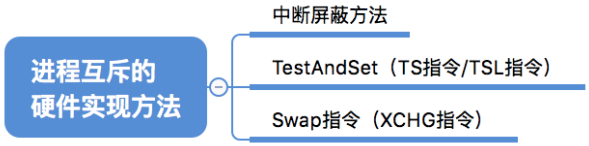
本节内容

进程互斥的硬件实现方法

知识总览



学习提示：
1. 理解各方法的原理
2. 了解各方法的优缺点

中断屏蔽方法

利用“开/关中断指令”实现（与原语的实现思想相同，即在某进程开始访问临界区到结束访问为止都不允许被中断，也就不能发生进程切换，因此也不可能发生两个同时访问临界区的情况）

```
...
关中断;
临界区;
开中断;
...
```

关中断后即不允许当前进程被中断，也必然不会发生进程切换

直到当前进程访问完临界区，再执行开中断指令，才有可能有别的进程上处理机并访问临界区

优点：简单、高效
缺点：不适用于多处理机；只适用于操作系统内核进程，不适用于用户进程（因为开/关中断指令只能运行在内核态，这组指令如果能让用户随意使用会很危险）

TestAndSet指令

简称 TS 指令，也有地方称为 TestAndSetLock 指令，或 TSL 指令
TSL 指令是用硬件实现的，执行的过程不允许被中断，只能一气呵成。以下是用C语言描述的逻辑

```
//布尔型共享变量 lock 表示当前临界区是否被加锁
//true 表示已加锁, false 表示未加锁
bool TestAndSet (bool *lock){
  bool old;
  old = *lock;  //old用来存放lock 原来的值
  *lock = true; //无论之前是否已加锁, 都将lock设为true
  return old;   //返回lock原来的值
}
```

```
//以下是使用 TSL 指令实现互斥的算法逻辑
while (TestAndSet (&lock)); //"上锁"并"检查"
临界区代码段...
lock = false;       //"解锁"
剩余区代码段...
```

若刚开始 lock 是 false，则 TSL 返回的 old 值为 false，while 循环条件不满足，直接跳过循环，进入临界区。若刚开始 lock 是 true，则执行 TLS 后 old 返回的值为 true，while 循环条件满足，会一直循环，直到当前访问临界区的进程在退出区进行“解锁”。
相比软件实现方法，TSL 指令把“上锁”和“检查”操作用硬件的方式变成了一气呵成的原子操作。
优点：实现简单，无需像软件实现方法那样严格检查是否会有逻辑漏洞；适用于多处理机环境
缺点：不满足“让权等待”原则，暂时无法进入临界区的进程会占用CPU并循环执行TSL指令，从而导致“忙等”。

Swap指令

有的地方也叫 Exchange 指令，或简称 XCHG 指令。
Swap 指令是用硬件实现的，执行的过程不允许被中断，只能一气呵成。以下是用C语言描述的逻辑

```
//Swap 指令的作用是交换两个变量的值
Swap (bool *a, bool *b) {
  bool temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
```

```
//以下是用 Swap 指令实现互斥的算法逻辑
//lock 表示当前临界区是否被加锁
bool old = true;
while (old == true)
  Swap (&lock, &old);
临界区代码段...
lock = false;
剩余区代码段...
```

逻辑上来看 Swap 和 TSL 并无太大区别，都是先记录下此时临界区是否已经被上锁（记录在 old 变量上），再将上锁标记 lock 设置为 true，最后检查 old，如果 old 为 false 则说明之前没有别的进程对临界区上锁，则可跳出循环，进入临界区。
优点：实现简单，无需像软件实现方法那样严格检查是否会有逻辑漏洞；适用于多处理机环境
缺点：不满足“让权等待”原则，暂时无法进入临界区的进程会占用CPU并循环执行TSL指令，从而导致“忙等”。

知识回顾与重要考点