互斥：硬件的实现

2012001010017　钱瑞和

目录

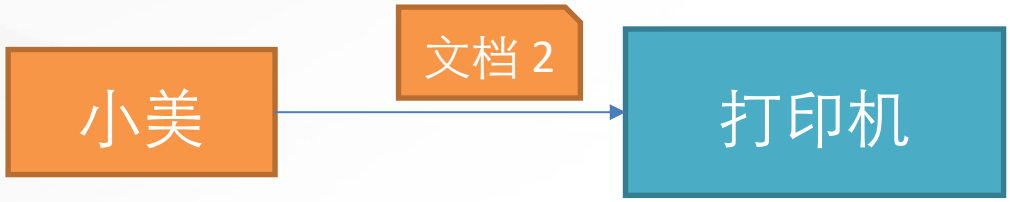
引子 - 一个关于打印机的故事

并发问题的硬件解决方案 - 单处理器

基本思想:
防止其他进程的执行

并发问题的硬件解决方案 - 单处理器

基本思想：
防止其他进程的执行



并发问题的硬件解决方案 - 单处理器

基本思想:
防止其他进程的执行

```
while （true）
{
        /* 禁用中断 */
        /* 临界区 */
        /* 启动中断 */
        /* 其余部分 */
}
```

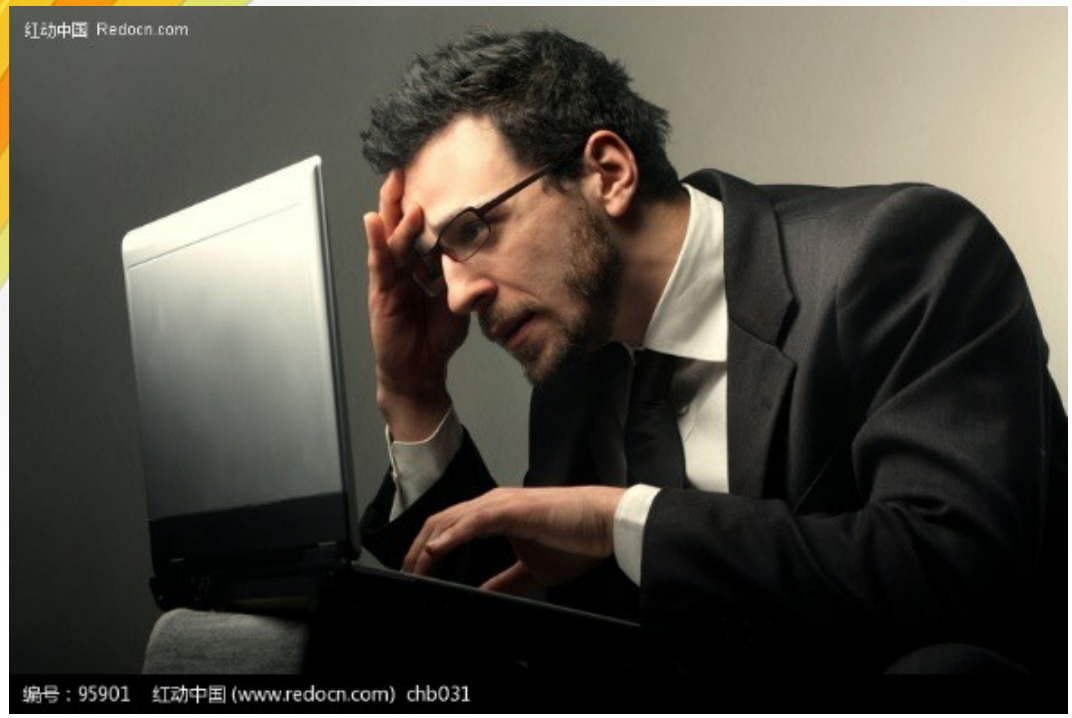
并发问题的硬件解决方案 - 单处理器

缺点

效率低下

不适用于多处理器

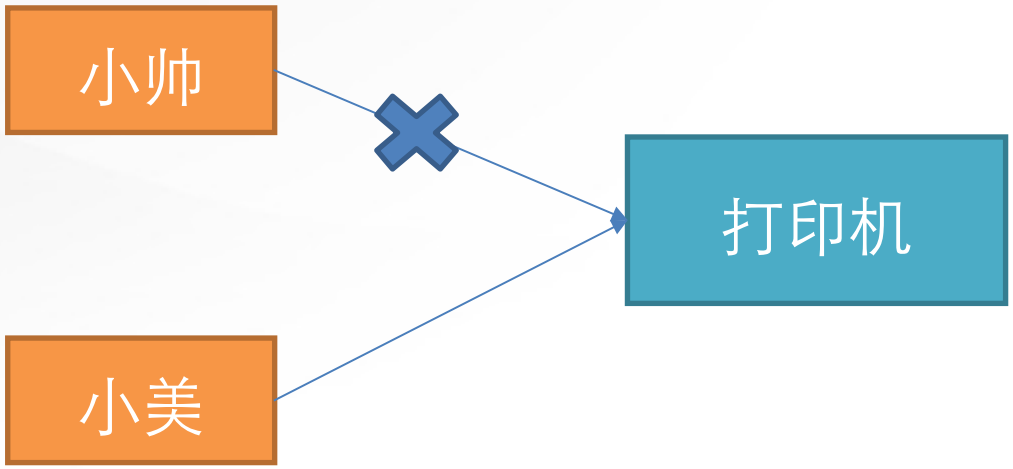
引子 - 一个关于共享打印机的故事



怎么保证我的数据在我使用的过程中时不被修改?

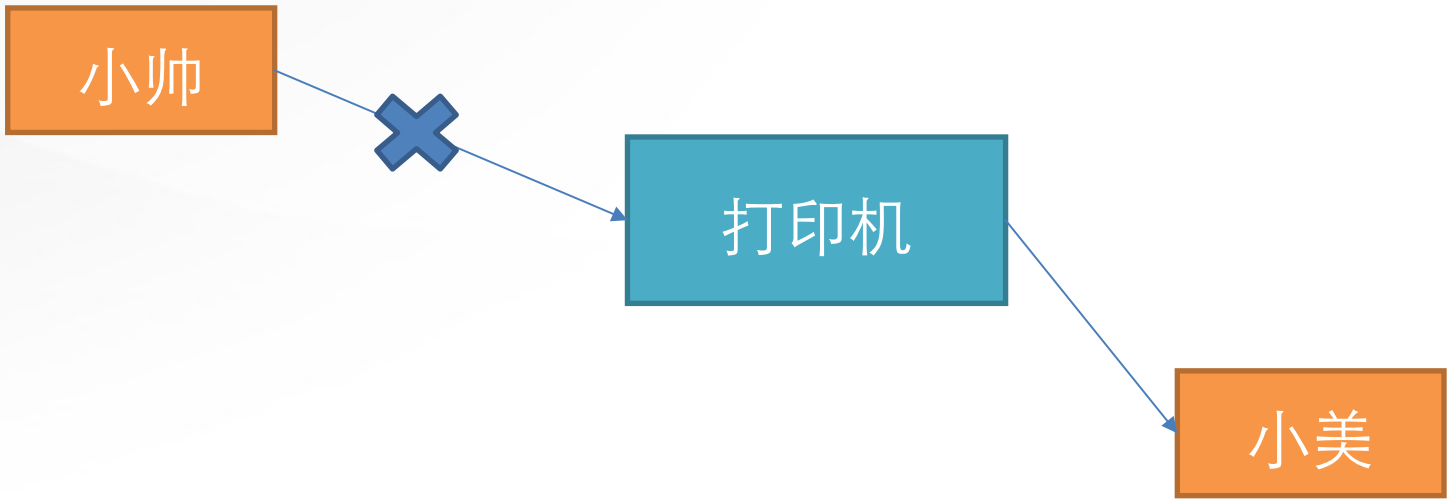
并发问题的硬件解决方案 - 多处理器

基本思想:
保证没有其他进程在我使用数据的时候修改数据



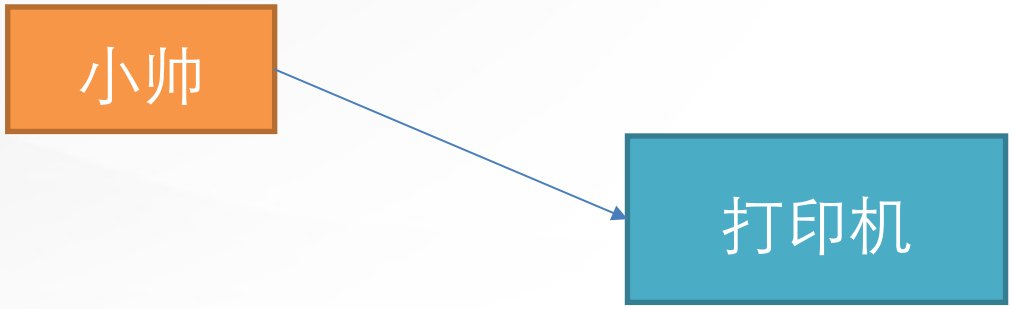
并发问题的硬件解决方案 - 多处理器

基本思想：
保证没有其他进程在我使用数据的时候修改数据



并发问题的硬件解决方案 - 多处理器

基本思想:
保证没有其他进程在我使用数据的时候修改数据



并发问题的硬件解决方案

基本思想:
保证没有其他进程在我使用数据的时候修改数据

方法一：设置“密码”锁住打印机

并发问题的硬件解决方案 - Compare-and-swap

基本思想：
保证没有其他进程在我使用数据的时候修改数据

```
int compare_and_swap(int *word, int testval, int newval)
{
        int oldval;
        oldval = *word;
        if(oldval == testval) *word = newval;
        // 相等所以内存没有改变，内存改为新值
        return oldval;
        // 返回旧内存值
}
```

并发问题的硬件解决方案 - Compare-and-swap

基本思想：
保证没有其他进程在我使用数据的时候修改数据

```
/* program mutualexclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
  {
    while (true) {
      while (compare_and_swap(bolt, 0, 1) == 1) ;
                /* critical section */;
      bolt = 0;
      /* remainder */;
    }
  }
void main()
 {
   bolt = 0;
   parbegin (P(1), P(2), ... ,P(n));
 }
```

并发问题的硬件解决方案 - Compare-and-swap

基本思想：
保证没有其他进程在我使用数据的时候修改数据

硬件支持：
X86- CMPXCHG

并发问题的硬件解决方案

基本思想:
保证没有其他进程在我使用数据的时候修改数据

方法二：设置“标志”锁住打印机

并发问题的硬件解决方案 - Exchange

基本思想：
保证没有其他进程在我使用数据的时候修改数据

```
int exchange(int register, int memory)
{
        int temp;
        temp = memory;
        memory = register;
        register = temp;
}
```

并发问题的硬件解决方案 - Exchange

基本思想：
保证没有其他进程在我使用数据的时候修改数据

```
/* program mutualexclusion */
int const n = /* number of processes**/;
int bolt;
void P(int i)
  {
     int keyi = 1;
     while (true) {
         do
            exchange (keyi, bolt)
         while (keyi != 0);
          /* critical section */;
          bolt = 0;
          /* remainder */;
     }
  }
void main()
  {
    bolt = 0;
    parbegin (P(1), P(2), ..., P(n));
  }
```

并发问题的硬件解决方案 - Exchange

基本思想：
保证没有其他进程在我使用数据的时候修改数据

硬件支持：
X86- XCHG

并发问题的硬件解决方案 - Test-and-set

基本思想：
保证没有其他进程在我使用数据的时候修改数据

```
int TestAndSet(int* lockPtr)
{
    int oldValue;
    oldValue = *lockPtr;
    *lockPtr = LOCKED;
    return oldValue;
}
```

并发问题的硬件解决方案 - Test-and-set

基本思想：
保证没有其他进程在我使用数据的时候修改数据

```
volatile int lock = 0;

void Critical()
 {
        while (TestAndSet(&lock) == 1);
                /* critical section */;
        lock = 0 // release lock when finished with the /* r
        /* remainder */;
}
```

并发问题的硬件解决方案 - Test-and-set

基本思想：
保证没有其他进程在我使用数据的时候修改数据

硬件支持：
X86- TSL

并发问题的硬件解决方案

基本思想:
保证没有其他进程在我使用数据的时候修改数据

方法三：设置“先后顺序”

并发问题的硬件解决方案 - Fetch-and-add

基本思想:
保证没有其他进程在我使用数据的时候修改数据

```
int FetchAndAdd(int* address, int turn)
 {
    int value = * address;
    * address := value + turn ;
    return value;
}
```

并发问题的硬件解决方案 - Fetch-and-add

基本思想：
保证没有其他进程在我使用数据的时候修改数据

硬件支持：
X86- XADD

并发问题的硬件解决方案

基本思想：
保证没有其他进程在我使用数据的时候修改数据

方法三：设置“标志”记录打印机是否被别人用过

并发问题的硬件解决方案 - Load-link/store-conditional

基本思想：
保证没有其他进程在我使用数据的时候修改数据

MIPS:

当使用 LL 指令从内存中读取一个字之后，处理器会记住 LL 指令的这次操作（会在 CPU 的寄存器中设置一个不可见的 bit 位），同时 LL 指令读取的地址，也会保存在处理器的寄存器中。

接下来的 SC 指令，会检查上次 LL 指令执行后的 RMW 操作是否是原子操作（即不存在其它对这个地址的操作），如果是原子操作，则 t 的值将会被更新至内存中，同时 t 的值也会变为 1，表示操作成功；反之，如果 RMW 的操作不是原子操作（即存在其它对这个地址的访问冲突），则 t 的值不会被更新至内存中，且 t 的值也会变为 0，表示操作失败。

优缺点

优点:
1、适用于在单处理器或共享内存的多处理器上　的任何数目的进程。
2、非常简单易于证明。
3、可支持多个临界区，每个临界区使用它自己的变量定义。

4、方便上层使用

优缺点

缺点:

1、 使用了忙等待。等待进入临界区的进程会继续消耗处理器时间。

2、 可能饥饿。选择哪一个等待进程是任意的，某些进程可能无限期地拒绝进入。

3、 可能死锁。进程 1 进入临界区后改变值，之后在进程 1 执行的过程中进程 1 被进程 2 抢占。如果进程 2 试图使用同一资源，由于进程 1 控制着该资源，进程 2 无法访问。所以进程 2 进入忙等待，但是进程 1 优先级低于进程 2，它将永远不会被调度执行。