作业1 XCHG的实现方法

吕佳鸿 10235501436

首先看一下XCHG指令

```
XCHG dest, src
```

在x86汇编中，XCHG是用来交换两个操作数的内容的指令 执行后 dest和src的内容会被交换

当交换两个寄存器的值时，通过临时寄存器，将 dest 的值暂存到一个临时寄存器中，然后将 src 的值复制到 dest，最后将临时寄存器中的值复制到 src

但在一些优化过的硬件实现中，交换寄存器的过程可能通过内部的绕线机制来实现，而不需要显式地使用临时寄存器，从而进一步加速。

而在交换内存和寄存器的值时，效率会有所降低。这是因为涉及到内存的读写操作，而内存访问的延迟通常比寄存器访问大得多。其实现可能如下：

```
temp := [mem]
[mem] := EAX
EAX := temp
```

在这里，[mem] 表示内存位置的内容。整个过程包括一次内存读和一次内存写，因此会涉及到缓存命中和内存层级等问题。

在实际应用中，XCHG可以用来实现自旋锁

```
spin_lock:
    MOV EAX, 1  ; EAX 中放置“锁定”状态
    XCHG EAX, [lock_var] ; 交换锁变量和 EAX 的值
    CMP EAX, 0  ; 检查锁变量之前是否是“未锁定”状态
    JNZ spin_lock ; 如果不为0（锁定状态），则继续自旋
```

作业2 Peterson算法

peterson算法的实现如下：

```
Pi:
flag[i] = ture; turn = j;
while (flag[j] && turn == j);
critical section;// 访问临界区
```

```
flag[i] = false;
remainder section;// 剩余区

Pj:
flag[j] = true; turn = i;
while (flag[i] && turn == i);
critical sectionl// 访问临界区
flag[j] = false;
remainder section;// 剩余区
```

Peterson算法是一种实现进程/线程间**互斥**访问**临界区**的算法。

关于互斥和临界区的定义如下：

`临界区` 一段代码，进程/线程在这段代码中进程将访问**共享资源**，当另外一个进程已在这段代码运行时，其他进程就不能在这段代码中运行。

`互斥` 当一个进程/线程在临界区访问共享资源时，其他进程/线程不能进入临界区访问任何其他共享资源的情形。

Peterson算法通过以下机制确保了两个进程在访问临界区时的互斥性：

互斥性：flag[i] 和 flag[j] 表示每个进程是否想进入临界区，而 turn 用于解决双方同时希望进入时的冲突。因为每个进程在试图进入时都会设置 turn 为对方，所以即使两个进程都试图进入临界区，最终也只有一个能先进入，而另一个会在 while 循环中等待。

无死锁：在这个算法中，如果一个进程已经进入临界区，另一个进程将一直在等待；但一旦第一个进程完成并退出，它会将 flag 设置为 false，从而保证另一个进程能顺利进入。

无饥饿（公平性）：Peterson算法的设计中，turn 的设置保证了如果两个进程都在等待，最后进入临界区的进程会将机会留给另一个进程。这样，避免了一个进程长时间占用临界区而让另一个进程一直无法进入的情况。

Peterson算法不需要原子（atomic）操作，是纯软件途径解决了互斥锁的实现。

peterson算法也可以扩充到N个进程实现，如下：

```
// initialization
level[N] = { -1 };     // current level of processes 0...N-1
waiting[N-1] = { -1 }; // the waiting process of each level 0...N-2

// code for process #i
for(l = 0; l < N-1; ++l) { // go through each level
    level[i] = l;
    waiting[l] = i;
    while(waiting[l] == i &&
          (there exists k ≠ i, such that level[k] ≥ l)) {
        // busy wait
    }
}

// critical section
```

```
level[i] = -1; // exit section
```